COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD

Publication number: KR20050086423 (A)
Publication date: 2005-08-30
Inventor(s): MATHUR CHAN DAN [US]; HELLENBACH SCOTT [US]; RAPP JOHN W [US]
Applicant(s): LOCKHEED CORP [US]
Classification:
- international: ***G06F9/30; G06F9/38; G06F9/445; G06F9/46; G06F15/78;*** **G06F9/30; G06F9/38; G06F9/445; G06F9/46; G06F15/76;** (IPC1-7): G06F9/46
- European: G06F9/38S4
Application number: KR20057007749 20050430
Priority number(s): US20030683929 20031009; US20030683932 20031009; US20030684053 20031009; US20030684057 20031009; US20030684102 20031009; US20020422503P 20021031

Also published as:
WO2004042560 (A2)
WO2004042560 (A3)
WO2004042574 (A2)
WO2004042574 (A3)
WO2004042569 (A2)
more >>

Abstract not available for KR 20050086423 (A)
Abstract of corresponding document: **WO 2004042560 (A2)**

A peer-vector machine includes a host processor and a hardwired pipeline accelerator. The host processor executes a program, and, in response to the program, generates host data, and the pipeline accelerator generates pipeline data from the host data. Alternatively, the pipeline accelerator generates the pipeline data, and the host processor generates the host data from the pipeline data. Because the peer-vector machine includes both a processor and a pipeline accelerator, it can often process data more efficiently than a machine that includes only processors or only accelerators. For example, one can design the peer-vector machine so that the host processor performs decision-making and non-mathematically intensive operations and the accelerator performs non-decision-making and mathematically intensive operations.; By shifting the mathematically intensive operations to the accelerator, the peer-vector machine often can, for a given clock frequency, process data at a speed that surpasses the speed at which a processor-only machine can process the data.

(19) KOREAN INTELLECTUAL PROPERTY OFFICE

KOREAN PATENT ABSTRACTS

(11) Publication number: 1020050086423 A
(43) Date of publication of application: 30.08.2005

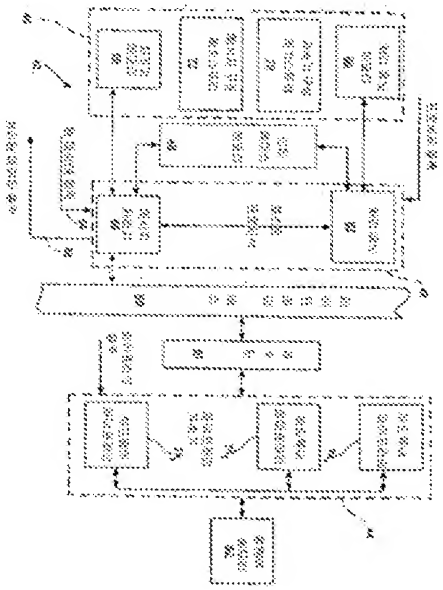
(21) Application number: 1020057007749
(22) Date of filing: 30.04.2005
(30) Priority: 2003 683929 US 09.10.2003
(51) Int. Cl: *G06F 9/46 (2006.01);*

(71) Applicant: LOCKHEED MARTIN CORPORATION
(72) Inventor: MATHUR CHAN DAN
HELLENBACH SCOTT
RAPP JOHN W.

(54) COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD

(57) Abstract:

A computing machine includes a first buffer and a processor coupled to the buffer. The processor executes an application, a first data-transfer object, and a second data-transfer object, publishes data under the control of the application, loads the published data into the buffer under the control of the first data-transfer object, and retrieves the published data from the buffer under the control of the second data- transfer object. Alternatively, the processor retrieves data and loads the retrieved data into the buffer under the control of the first data- transfer object, unloads the data from the buffer under the control of the second data-transfer object, and processes the unloaded data under the control of the application. Where the computing machine is a peer- vector machine that includes a hardwired pipeline accelerator coupled to the processor, the buffer and data-transfer objects facilitate the transfer of data between the application and the accelerator.



This Facsimile First Page has been artificially created from the Korean Patent Abstracts CD Rom

(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。 Int. Cl.[7]
G06F 9/46

(11) 공개번호 10-2005-0086423
(43) 공개일자 2005년08월30일

(21) 출원번호 10-2005-7007749
(22) 출원일자 2005년04월30일
번역문 제출일자 2005년04월30일
(86) 국제출원번호 PCT/US2003/034669
국제출원일자 2003년10월31일
(87) 국제공개번호 WO 2004/042574
국제공개일자 2004년05월21일

(30) 우선권주장
10/683,929 2003년10월09일 미국(US)
10/683,932 2003년10월09일 미국(US)
10/684,053 2003년10월09일 미국(US)
10/684,057 2003년10월09일 미국(US)
10/684,102 2003년10월09일 미국(US)
60/422,503 2002년10월31일 미국(US)

(71) 출원인 록히드 마틴 코포레이션
미국 버지니아주 20110, 매나사스, 콩원 드라이브 9500, 메일 드롭 043, 빌딩 400

(72) 발명자 에튜, 웨단
미국 버지니아 20109, 매나사스, 프라이베이츠 코트 11162
웰렌바흐, 스콧
미국 버지니아 20106, 아메스빌, 루아일 리지 드라이브 15381
웹, 존 더블류
미국 버지니아 20110, 매나사스, 리버 크레스트 로드 9350

(74) 대리인 전영일
양주석

심사청구 : 없음

(54) 향상된 컴퓨팅 아키텍처를 갖는 컴퓨팅 머신 및 관련시스템 및 방법

요약

파이프라인 가속기에는 메모리 및 상기 메모리와 결합된 하드와이어드-파이프라인 회로가 포함되어 있다. 상기 하드와이어드-파이프라인 회로는 데이터를 수신하고, 이 데이터를 메모리에 로드하고, 상기 메모리에서 이 데이터를 검색하고, 상기 검색된 데이터를 처리하고, 상기 처리된 데이터를 외부 소스로 제공하도록 동작 가능하다. 추가로 또는 선택적으로, 상기 하드와이어드-파이프라인 회로는 데이터를 수신하고, 상기 수신된 데이터를 처리하고, 상기 처리된 데이터를 메모리에 로드하고, 상기 메모리에서 상기 처리된 데이터를 검색하고, 상기 검색된 처리된 데이터를 외부 소스로 제공하도록 동작 가능하다. 파이프라인 가속기가 피어-넥터 머신의 일부로서의 프로세서와 결합하면, 상기 메모리는 상기 하드와이어드-파이프라인 회로와 상기 프로세서가 실행하는 애플리케이션 사이에서 - 단방향 또는 양방향으로 - 데이터 전송을 용이하게 한다.

대표도

도 3

색인어

파이프라인, 파이프라인 가속기, 하드와이어드-파이프라인

명세서

기술분야

(우선권 주장)

본 출원은 참고문헌으로서 통합되는 2002년 10월 31일 출원된 미국 가출원 제60/422,503호의 우선권을 주장한다.

(관련된 출원과의 상호참조)

본 출원은 발명의 명칭이 "향상된 컴퓨팅 아키텍처 및 관련 시스템 및 방법" 인 미국 특허출원 제10/684,102호, 발명의 명칭이 "향상된 컴퓨팅 아키텍처를 위한 파이프라인 가속기 및 관련된 시스템 및 방법" 인 미국 특허출원 제 10/683,929호, 발명의 명칭이 "프로그램가능한 회로 및 관련 컴퓨팅 머신 및 방법" 인 미국 특허출원 제10/684,057호, 및 발명의 명칭이 "다중 파이프라인 유닛을 가지는 파이프라인 가속기 및 관련 컴퓨팅 머신 및 방법" 인 미국 특허출원 제 10/684,932호와 관련이 되어 있으며, 상기 미국 특허출원 발명은 모두 2003년 10월 9일 동일한 권리자에 의해 출원되었으며, 본 명세서에 참고문헌으로 통합된다.

배경기술

상대적으로 짧은 시간에서 상대적으로 대용량의 데이터를 처리하기 위한 일반적인 컴퓨팅 아키텍처(computing architecture)에는 부하를 분배하는 다중 상호접속 프로세서(multiple interconnected processor)가 포함되어 있다. 처리 부하를 분배함으로써, 이들 다중 프로세서들은 주어진 클록 주파수에서 하나의 프로세서가 할 수 있는 것 보다 더욱 빨리 데이터를 처리할 수 있다. 예를 들어, 프로세서 각각은 데이터의 일부를 처리 하거나 또는 처리되는 알고리즘 일부를 실행할 수 있다.

도 1은 다중-프로세서 아키텍처를 갖는 종래의 컴퓨팅 머신(computing machine)(10)의 개략적인 블록 다이어그램이다. 머신(10)에는 마스터 프로세서(12) 및 버스(16)를 통해 상기 마스터 프로세서와 상호 통신을 하는 코프로세서(14_1-14_n), 원격 장치(도 1에는 도시하지 않음)로부터 원 데이터(raw data)를 수신하는 입력 포트(18), 및 처리된 데이터를 상기 원격 소스로 제공하는 출력 포트(20)가 포함되어 있다. 상기 머신(10)에는 또한 마스터 프로세서(12)용 메모리(22), 코프로세서(14_1-14_n)용 메모리(24_1-24_n), 및 상기 버스(16)를 통해 마스터 프로세서와 코프로세서가 공유하는 메모리(26)도 포함되어 있다. 메모리(22)는 마스터 프로세서(12)를 위한 프로그램 및 작업 메모리 모두의 역할을 하고, 메모리(24_1-24_n) 각각은 코프로세서(14_1-14_n) 각각을 위한 프로그램 및 작업 메모리 모두의 역할을 한다. 공유 메모리(26)는 마스터 프로세서(12)와 코프로세서(14)가 그들 사이의 데이터를 포트(18 및 20)을 통해 각각 원격 장치로/장치로부터 전달할 수 있게 해준다. 마스터 프로세서(12) 및 코프로세서(14)는 또한 머신(10)이 원 데이터를 처리하는 속도를 제어하는 공통의 클록 신호를 수신하기도 한다.

일반적으로, 컴퓨팅 머신(10)은 마스터 프로세서(12)와 코프로세서(14) 간의 원 데이터 처리를 효과적으로 분배한다. 소나 어레이(sonar array)와 같은 원격 소스(도 1에는 도시하지 않음)는 포트(18)를 통해 원 데이터를 상기 원 데이터를 위한 선입선출(FIFO) 버퍼(도시하지 않음)로서 작동하는 공유 메모리(26)의 일부에 로드(load)한다. 마스터 프로세서(12)는 버스(16)를 통해 메모리(16)로부터 원 데이터를 검색하고, 마스터 프로세서와 코프로세서(14)가 상기 원 데이터를 처리하고, 버스(16)를 통해 필요한 만큼 그들사이에서 데이터를 전달한다. 마스터 프로세서(12)는 처리된 데이터를 공유 메모리(26)에 정의되어 있는 다른 FIFO 버퍼(도시하지 않음)에 로드하고, 원격 소스가 포트(20)를 통해 이 FIFO 로부터 상기 처리된 데이터를 검색한다.

다른 연산의 예에서, 컴퓨팅 머신(10)은 원 데이터를 처리하는데 있어서 상기 원 데이터상에서 각각의 연산에 n+1을 순차적으로 수행하여 처리하는데, 이 연산은 패스트 푸리에 변환(FFT)과 같은 처리 알고리즘을 함께 구성한다. 보다 특별하게는, 머신(10)은 마스터 프로세서(12)와 코프로세서(14)로부터 데이터-처리 파이프라인을 형성한다. 주어진 클록 신호의 주파수를 위해, 그러한 파이프라인은 종종 머신(10)이 하나의 프로세서만 가지는 머신보다 더욱 빠르게 원 데이터를 처리할 수 있게 한다.

메모리(26)내의 원-데이터 FIFO(도시하지 않음)로부터 원 데이터를 검색한 후, 마스터 프로세서(12)는 삼각 함수와 같은 제1 연산을 상기 원 데이터상에 수행한다. 이 연산은 프로세서(12)가 메모리(26) 내부에 정의된 제1-결과 FIFO(도시하지 않음)내에 저장하는 첫번째 결과를 산출해 낸다. 일반적으로, 프로세서(12)는 메모리(12) 내에 저장된 프로그램을 수행하며, 그 프로그램의 제어하에 상기 설명된 연산들을 수행한다. 프로세서(12)는 또한 메모리(22)를 작업 메모리로 사용하여 프로세서가 상기 제1 연산의 중간 시간에 생성되는 데이터를 일시적으로 저장하기도 한다.

다음으로, 상기 메모리(26)내의 제1-결과 FIFO(도시하지 않음)로부터 첫번째 결과를 검색한 후, 코프로세서(14)는 로그 함수와 같은 두번째 연산을 상기 첫번째 결과상에 수행한다. 이 두번째 연산은 코프로세서(14_1)가 메모리(26) 내부에 정의된 제2-결과 FIFO(도시하지 않음)내에 저장하는 두번째 결과를 산출해 낸다. 일반적으로, 코프로세서(14_1)는 메모리(24_1) 내에 저장된 프로그램을 수행하며, 그 프로그램의 제어하에 상기 설명된 연산들을 수행한다. 코프로세서(14)는 또한 메모리(24_1)를 작업 메모리로 사용하여 코프로세서가 상기 제2 연산의 중간 시간에 생성되는 데이터를 일시적으로 저장하기도 한다.

그리고 나서, 코프로세서(24_2-24_n)는 상기 두번째-(n-1)번째 상에 세번째-n번째 연산을 순차적으로 수행하여 상기 코프로세서(24_1)를 위해 상기 설명한 것과 유사한 방법의 결과를 가져온다.

코프로세서(24_n)에 의해 수행되는 n번째 연산은 최종 결과, 즉 처리된 데이터를 산출한다. 코프로세서(24_n)는 메모리(26) 내부에 정의된 처리된-데이터 FIFO(도시하지 않음)로 이 처리된 데이터를 로드(load)하고, 원격 장치(도 1에는 도시하지 않음)가 이 FIFO로부터 상기 처리된 데이터를 검색한다.

마스터 프로세서(12)와 코프로세서(14)가 처리 알고리즘의 다른 연산을 동시에 수행하기 때문에, 컴퓨팅 머신(10)은 서로 다른 연산을 순차적으로 수행하는 하나의 프로세서를 갖는 컴퓨팅 머신 보다도 원 데이터를 보다 빨리 처리할 수 있기도 하다. 특히, 하나의 프로세서는 원 데이터의 앞 세트상에 모든 n+1 연산을 수행할 때 까지는 원 데이터의 새로운 세트를 검색할 수 없다. 그러나, 상기 언급한 파이프라인 기술을 이용해서, 마스터 프로세서(12)는 오직 첫번째 연산을 수행한 후 원 데이터의 새로운 세트를 검색할 수 있다. 따라서, 주어진 클록 주파수를 위해, 이 파이프라인 기술은 머신(10)이 원 데이터를 처리하는데 있어서 하나의 프로세서 머신(도 1에는 도시하지 않음)과 비교할 때 대략 n+1 팩터 만큼 원 데이터를 처리하는 속도를 증가시킬 수 있다.

선택적으로, 컴퓨팅 머신(10)은 원 데이터상에, FFT 와 같은 처리 알고리즘의 경우에 n+1을 동시에 수행함으로써 병렬로 원 데이터를 처리할 수도 있다. 즉, 알고리즘에 앞서의 실시예에서 상기 언급한 바와 같은 n+1 순차적 연산이 포함되어 있다면, 마스터 프로세서(12)와 코프로세서(14) 각각은 모든 n+1 연산을 원 데이터 각각의 세트상에서 수행한다. 따라서, 주어진 클록 주파수를 위해서는, 상기 설명한 파이프라인 기술과 같이, 이러한 병렬-처리 기술은 머신(10)이 하나의 프로세서 머신(도 1에는 도시하지 않음)과 비교할 때 대략 n+1 팩터 만큼 원 데이터 처리 속도를 증가시킬 수 있다.

불행하게도, 비록 컴퓨팅 머신(10)이 하나의 프로세서 컴퓨터 머신(도 1에는 도시하지 않음)보다 빨리 데이터를 처리할 수는 있으나, 머신(10)의 데이터-처리 속도가 종종 프로세서 클록의 주파수보다 적어지곤 한다. 특히, 컴퓨팅 머신(10)의 데이터-처리 속도는 마스터 프로세서(12)와 코프로세서(14)가 데이터를 처리하는데 요구하는 시간에 의해 제한된다. 간략히 하기 위해, 이 속도 제한의 한 예를 마스터 프로세서(12)를 참고하여 설명하는데, 이 설명은 코프로세서(14)에도 적용됨을 이해할 수 있을 것이다. 앞에서 설명한 바와 같이, 마스터 프로세서(12)는 프로세서를 제어하여 데이터를 원하는 방식으로 조작하도록 제어하는 프로그램을 실행한다. 이 프로그램에는 프로세서(12)가 실행하는 일련의 명령이 포함되어 있다. 불행하게도, 프로세서(12)는 일반적으로 하나의 명령을 수행하는데 다중 클록 사이클을 필요로 하는데, 종종 다중 명령을 수행하여 데이터의 하나의 값을 처리해야 한다. 예를 들어, 프로세서(12)가 제1 데이터 값(A)(도시하지 않음)과 제2 데이터 값(B)(도시하지 않음)을 곱하는 경우를 가정한다. 제1 클록 사이클 동안, 프로세서(12)는 메모리(22)로부터 여러개의 명령을 검색한다. 제2 및 제3 클록 사이클 동안, 프로세서(12)는 메모리(22)로부터 A와 B를 각각 검색한다. 제4 클록 사이클 동안, 프로세서(12)는 A와 B를 곱하고, 제5 클록 사이클 동안, 그 결과물을 메모리(22 또는 26)에 저장하거나 또는

그 결과물을 원격 장치(도시하지 않음)로 제공한다. 이것은 최선의 시나리오인데, 그 이유는 많은 경우에서, 프로세서(12)는 카운터를 초기화(initializing) 및 닫는(closing) 경우와 같이 오버헤드 태스크(overhead task)를 위한 추가의 클록 사이클을 요구하기 때문이다. 그러므로, 프로세서(12)는 A와 B를 처리하기 위해서는 5개의 클록 사이클, 또는 데이터 값 당 평균 2.5개의 클록 사이클을 필요로 한다.

따라서, 컴퓨팅 머신(10)이 데이터를 처리하는 속도는 종종 마스터 프로세서(12) 및 코프로세서(14)를 구동하는 클록의 주파수보다 크게 낮아지곤 한다. 예를 들어, 프로세서(12)가 1.0 기가헤르쯔(GHz)에서 클록되지만 데이터 값 당 평균 2.5 클록 사이클을 요구한다면, 유효 데이터-처리 속도는 (1.0GHz)/2.5=4GHz 가 될 것이다. 이 유효 데이터-처리 속도는 종종 초당 연산 단위로 특정되곤 한다. 그러므로, 1.0GHz 의 클록 속도를 위해서는, 프로세서(12)는 초당 0.4 기가연산(Gops)의 데이터-처리 속도로 레이트(rate)되어야 한다.

도 2는 프로세서가 주어진 클록 주파수 및 종종 파이프라인이 클록되는 레이트와 거의 동일한 레이트에서 할 수 있는 것보다 더 빠른 일반적인 데이터를 처리할 수 있는 하드와이어드(hardwired) 데이터 파이프라인(30)의 블록 다이어그램이다. 파이프라인(30)에는 각각 실행 프로그램 명령 없이 각각의 데이터상의 개별적 연산을 각각 수행하는 연산자 회로(32_1-32_n)가 포함되어 있다. 즉, 원하는 연산이 회로(32)로 "번 되는(burned in)" 것으로 프로그램 명령 없이도 자동적으로 연산을 실행하는 것이다. 실행 프로그램 명령과 관련된 오버헤드를 제어함으로써, 파이프라인(30)은 종종 주어진 클록 주파수를 위해 할 수 있는 것보다 더 많은 연산을 수행할 수 있다.

예를 들어, 파이프라인(30)은 프로세서가 주어진 클록 주파수를 위해 할 수 있는 것 보다 더 빠른 다음과 같은 수식을 풀 수 있다.

$$Y(X_k)=(5X_k+3)2^{X_k}$$

여기서, Xk는 입력의 원 데이터 값을 나타낸다. 이 예에서, 연산자 회로(32_1)는 5Xk 를 계산하는 곱셈기이고, 회로(32_2)는 5Xk + 3 을 계산하는 가산기이며, 회로(32_n)(n=3)는 (5Xk + 3)2^{Xk} 를 계산하는 곱셈기이다.

제1 클록 사이클 k=1 동안, 회로(32_1)는 데이터 값(X_1)을 수신하고 여기에 5를 곱해 $5X_1$ 을 생성한다.

제2 클록 사이클 k=2 동안, 회로(32_2)는 회로(32_1)로부터 $5X_1$을 수신하고, 3을 더해서 $5X_1$ + 3 을 생성한다. 또한, 상기 제2 클록 사이클 동안, 회로(32_1)는 $5X_2$ 를 생성한다.

제3 클록 사이클 k=3 동안, 회로(32_3)는 회로(32_2)로부터 $5X_1$ + 3 을 수신하고, 2^{X1} (x_1 만큼 유효하게 $5X_1$ + 3 왼쪽 시프트)을 곱하여 첫번째 결과(5X_1 + 3)2^{X1} 을 생성한다. 또한, 제3 클록 사이클 동안, 회로(32_1)는 $5X_3$ 를 생성하고 회로(32_2)는 $5X_2$ + 3 을 생성한다.

파이프라인(30)은 이러한 방식으로 모든 원 데이터 값이 처리될 때 까지 연속적인 원 데이터 값(X_k)을 계속 처리한다.

따라서, 원 데이터 값(X_1)을 수신한 후 두 개의 클록 사이클의 지연 - 이 지연을 파이프라인(30)의 레이턴시(latency)라고 부르곤 함 - 상기 파이프라인은 결과 (5X_1 + 3)2^{X1} 을 생성하고, 그 후에 하나의 결과 - 예를 들어, (5X_2+ 3)2^{X2}, (5X_3+ 3)2^{X3},...5X_n+ 3)2^{Xn}, 를 각각의 클록 사이클을 생성한다.

상기 레이턴시를 무시하면, 파이프라인(30)은 클록 속도와 동일한 데이터-처리 속도를 갖는다. 비교를 하면, 마스터 프로세서(12)와 코프로세서(14)가 상기 예에서와 같은 클록 속도의 0.4배의 데이터-처리 속도를 갖는다고 가정하면, 파이프라인(30)은 주어진 클록 속도를 위해 범용형 머신(10)(도 1)보다 2.5배 빨리 데이터를 처리할 수 있다.

도 2를 계속 참조하면, 설계자는 파이프라인(30)을 필드-프로그램가능한 게이트 어레이(FPGA)와 같은 프로그램가능한 로직 IC(PLIC)에서 수행하도록 선택하기도 하는데, 그 이유는 PLIC 는 주문형 반도체(ASIC)보다 나은 디자인 및 변형 유연성 가진다. PLIC 내부에서 하드와이어드 접속을 구성하기 위해서는, 설계자는 단지 PLIC 내부에 배치된 상호접속-구성 레지스터를 미리 결정된 이진 상태(binary state)로 설정하기만 하면 된다. 이들 이진 상태 모두의 조합을 "펌웨어(firmware)"라고 부르곤 한다. 일반적으로, 설계자는 이 펌웨어를 PLIC 와 결합된 비휘발성 메모리(도 2에 도시하지 않음)에 프로그램한다. 누군가가 PLIC 를 "켜면(turn on)", PLIC는 상기 메모리로부터 펌웨어를 상기 상호접속-구조 레지스터로 다운로드한다. 그러므로, PLIC 의 기능을 변경시키기 위해서는, 설계자는 단지 펌웨어를 수정하면 되고 PLIC 로 하여금 그 수정된 펌웨어를 상호접속-구조 레지스터로 다운로드하게 하면 된다. 이것은 단지 펌웨어를 수정하는 것에 의해 PLIC를 수정할 수 있다는 것은 시제품화 단계 동안 및 "필드 내에서" 파이프라인(30)의 업그레이드를 위해 특히 유용하다.

불행하게도, 하드와이어드 파이프라인(30)은 중요한 결정 형성(decision making)을 특히 필요로 하는 모든 알고리즘을 실행하지 못한다. 프로세서는 비교가능한 길이의 연산 명령(예를 들어, "A+B")를 실행할 수 있는 정도로 거의 빠르게 일반적으로 결정-형성 명령(예를 들어, "A이면 B로 가고, 그렇지 않으면 C로 가고"와 같은 명령)를 실행할 수 있다. 그러나, 비록 파이프라인(30)이 상대적으로 간단한 결정(예를 들어, "A>B?")을 할 수 있어도, 일반적으로는 상대적으로 복잡한 결정(예를 들어, "A이면 B로 가고, 그렇지 않으면 C로 가고")을 실행할 수 없다. 그리고, 비록 그러한 복잡한 결정을 실행하기 위해 파이프라인(30)을 디자인할 수 있다 하여도, 요구되는 회로의 크기와 복잡성은 종종 설계를 불가능하게 만드는데, 특히 여러개의 서로 다른 복잡한 결정을 포함하는 알고리즘인 경우 그러하다.

따라서, 프로세서는 중요한 결정 형성을 요구하는 애플리케이션 내에서 종종 사용되며, 하드와이어드 파이프라인은 결정 형성이 거의 없는 또는 전혀 없는 "수치처리(number crunching)" 애플리케이션으로 제한되곤 한다.

더욱이, 아래에 설명한 바와 같이, 도 2의 파이프라인(30)과 같은 하드와이어드 파이프라인, 특히 여러개의 PLIC를 포함하는 파이프라인(30)을 설계/수정하는 것 보다는 도 1의 컴퓨팅 머신(10)과 같은 프로세서-기반 컴퓨팅 머신을 설계/수정하는 것이 훨씬 쉽다.

프로세서와 그 주변장치들(예를 들어, 메모리)과 같은 컴퓨팅 구성요소들은 일반적으로 이 구성요소들의 상호접속을 촉진하여 프로세서-기반 컴퓨팅 머신을 형성하기 위한 산업-표준 통신 인터페이스를 포함한다.

특히, 표준 통신 인터페이스에는 두 개의 계층(layer)이 포함되는데, 물리 계층과 서비스 계층이다. 물리 계층에는 회로 및 이 회로의 연산 파라메터 및 인터페이스를 형성하는 대응 회로 상호접속이 포함되어 있다. 예를 들어, 물리 계층에는 구성요소를 버스와 연결하는 핀, 이 핀으로부터 데이터를 래치(latch)하는 버퍼, 및 이 핀상에서 신호를 구동시키는 구동기가 포함되어 있다. 상기 연산 파라메터에는 상기 핀이 수신하는 데이터 신호의 허용가능한 전압 범위, 데이터를 기록 및 판독하는 신호 타이밍, 및 지지된 연산 모드(예를 들어, 버스트 모드, 페이지 모드)가 포함되어 있다. 종래의 물리 계층에는 트랜지스터-트랜지스터 논리(TTL) 및 램버스(RAMBUS)가 포함된다.

서비스 계층에는 컴퓨팅 구성요소가 데이터를 전송하는 프로토콜이 포함된다. 이 프로토콜은 데이터의 포맷 및 상기 구성요소가 포맷된 데이터를 송수신하는 방식을 정의한다. 종래의 통신 프로토콜에는 파일-전송 프로토콜(FTP) 및 TIP/IP(확장)가 포함된다.

따라서, 산업-표준 통신 인터페이스를 갖는 제조자 및 다른 일반적인 설계 컴퓨팅 구성요소들로 인해서, 그러한 구성요소의 인터페이스를 일반적인 설계로 할 수 있고 이것을 상대적으로 적은 노력으로 다른 컴퓨팅 구성요소들과 상호접속시킬 수 있다. 이것은 설계자에게 대부분의 시간을 컴퓨팅 머신의 다른 부분들을 설계하는데 소비하게 만들고, 구성요소들을 추가하거나 없애는 것을 통해 머신을 쉽게 수정할 수 있게 한다.

산업-표준 통신 인터페이스를 지원하는 컴퓨팅 구성요소를 설계하는 것은 설계 라이브러리(design library)로부터 현존하는 물리-계층 설계를 사용하여 설계 시간을 짧아하게 해 준다. 이것은 또한 구성요소들을 제조품인 컴퓨팅 구성요소들과 쉽게 접속할 수 있게 해주기도 한다.

공통의 산업-표준 통신 인터페이스를 지원하는 컴퓨팅 구성요소를 사용하여 컴퓨팅 머신을 설계하는 것은 설계자로 하여금 시간과 노력을 줄여주면서 구성요소들을 상호접속할 수 있게 해 준다. 이들 구성요소들이 공통의 인터페이스를 지원하기 때문에, 설계자는 설계 노력을 거의 들이지 않고 시스템 버스를 통해 이들을 상호 접속할 수 있다. 지원되는 인터페이스가 산업 표준이기 때문에, 설계자는 머신을 쉽게 수정할 수 있다. 예를 들어, 설계자는 다른 구성요소 및 주변장치들을 머신에 추가하여 시스템 설계를 발전시켜 나갈 수 있으며, 또는 차세대 구성요소들을 쉽게 추가/설계하여 기술 발전을 이

를 수 있다. 더욱이, 구성요소들이 공통의 산업-표준 서비스 계층을 지원하기 때문에, 설계자는 컴퓨팅 머신의 소프트웨어로 대응하는 프로토콜을 실행하는 현존하는 소프트웨어 모듈을 합체시킬 수 있다. 따라서, 설계자는 인터페이스 설계가 이미 적절하게 필수적이기 때문에 거의 노력을 들이지 않고 구성요소들을 접속시킬 수 있어서 머신의 특정 기능을 수행하게 하는 머신의 일부(예를 들어, 소프트웨어)를 설계하는 데 주력할 수 있다.

그러나, 불행하게도, 도 2의 파이프라인(30)과 같은 하드와이어드 파이프라인을 형성하는데 사용되는 PLIC 등과 같은 구성요소를 위한 알려진 산업-표준 통신 인터페이스는 없다.

따라서, 여러개의 PLIC 를 갖는 파이프라인을 설계하기 위해서, 설계자는 "스크래치(scratch)로부터" PLIC 간의 통신 인터페이스를 설계하고 디버깅하는데 상당한 시간과 노력을 들여야 한다. 일반적으로, ad hoc 통신 인터페이스는 PLIC 간에서 전달되는 데이터의 파라메터에 따라 달라진다. 비슷하게, 프로세서와 접속되는 파이프라인을 설계하기 위해서는, 설계자는 스크래치로부터 파이프라인과 프로세서간의 통신 인터페이스의 서비스 계층을 설계하고 디버깅하는데 상당한 시간과 노력을 들여야 한다.

비슷하게, PLIC 을 추가하는 것으로 파이프라인을 수정하기 위해서는, 설계자는 일반적으로 추가된 PLIC와 현재의 PLIC 사이의 통신 인터페이스를 설계하고 디버깅하는데 상당한 시간과 노력을 들이게 된다. 그리고, 프로세서를 추가하는 것으로 파이프라인을 수정하려면, 또는, 파이프라인을 추가하는 것으로 컴퓨팅 머신을 수정하려면, 설계자는 파이프라인과 프로세서간의 통신 인터페이스를 설계하고 디버깅하는데 상당한 시간과 노력을 들여야 한다.

그러므로, 도 1 및 도 2를 참고하면, 다수의 PLIC 를 접속하고 파이프라인과 프로세서의 접속 어려움으로 인해, 설계자는 컴퓨팅 머신을 설계하는 경우 상당한 트레이드오프(tradeoff)에 직면하곤 한다. 예를 들어, 프로세서-기반 컴퓨팅 머신을 가지고는, 설계자는 복잡한 결정-형성 가능성을 위한 트레이드 수-코딩성 속도 및 설계/수정 유연성에 집중하게 된다. 반대로, 하드와이어드 파이프라인-기반 컴퓨팅 머신을 가지고는, 설계자는 수-코딩성 속도를 위한 트레이드 복잡성-결정-형성 가능성 및 설계/수정에 집중하게 된다. 더욱이, 다수의 PLIC 를 접속하는데의 어려움으로 인해, 소수의 PLIC 를 가지는 파이프라인-기반 머신을 설계하는 것이 불가능하기도 하다. 그 결과, 실제적인 파이프라인-기반 머신은 제한된 기능을 갖곤 한다. 그리고, 프로세서와 PLIC 와의 접속 어려움으로 인해서, 하나의 PLIC 이상과 프로세서와의 접속이 불가능하기도 하다. 그 결과, 프로세서와 파이프라인을 합침으로써 얻어지는 이익이 적다.

따라서, 하드와이어드-파이프라인-기반 머신의 수-코딩성 속도를 가지고 프로세서-기반 머신의 결정-형성 가능성을 결합시킬 수 있는 새로운 컴퓨팅 아키텍처 요구가 있어 왔다.

발명의 상세한 설명

(요약)

본 발명의 한 실시예에서, 컴퓨팅 머신에는 제1 버퍼 및 상기 버퍼와 결합된 프로세서가 포함되어 있다. 상기 프로세서는 애플리케이션, 제1 데이터-전송 오브젝트, 및 제2 데이터-전송 오브젝트를 실행하고, 상기 애플리케이션의 제어하에 데이터를 발행하고, 상기 제1 데이터-전송 오브젝트의 제어하에 상기 발행된 데이터를 상기 버퍼로 로드하고, 그리고 상기 제2 데이터-전송 오브젝트의 제어하여 상기 버퍼로부터 상기 발행된 데이터를 검색하도록 동작 가능하다.

본 발명의 다른 실시예에 따르면, 상기 프로세서는 데이터를 검색하고, 상기 제1 데이터-전송 오브젝트의 제어하에 상기 검색된 데이터를 상기 버퍼로 로드하고, 상기 제2 데이터-전송 오브젝트의 제어하에 상기 버퍼로부터 상기 데이터를 언로드하고, 그리고 상기 애플리케이션의 제어하에 상기 언로드된 데이터를 처리하도록 동작 가능하다.

컴퓨팅 머신은 피어-벡터(peer-vector) 머신으로서, 상기 프로세서와 결합되어 있는 하드와이어드 파이프라인 가속기를 포함하고 있으며, 상기 버퍼 및 데이터-전송 오브젝트는 상기 애플리케이션과 상기 가속기 사이의 - 양방향 또는 단방향으로 - 데이터 전송을 촉진한다.

도면의 간단한 설명

도 1은 종래의 다중-프로세서 아키텍처를 갖는 컴퓨팅 머신의 블럭 다이어그램이고,

도 2는 종래의 하드와이어드 파이프라인의 블럭 다이어그램이고,

도 3은 본 발명의 일 실시예에 따른 피어-벡터 아키텍처를 갖는 컴퓨팅 머신의 개략적인 블록 다이어그램이고,

도 4는 본 발명의 일 실시예에 따른 도 3의 호스트 프로세서의 기능 블록 다이어그램이고,

도 5는 본 발명의 일 실시예에 따른 도 4의 상기 데이터-처리 애플리케이션 및 파이프라인 버스간의 데이터 전송 경로의 기능 블록 다이어그램이고,

도 6은 본 발명의 일 실시예에 따른 도 4의 가속기 에의 관리자 및 파이프라인 버스간의 데이터-전송 경로의 기능 블록 다이어그램이며,

도 7은 본 발명의 일 실시예에 따른 도 4의 가속기 구성 관리자 및 파이프라인 버스간의 데이터-전송 경로의 기능 블록 다이어그램이다.

실시예

(상세한 설명)

도 3은 본 발명의 일 실시예에 따른 피어-벡터 아키텍처를 갖는 컴퓨팅 머신(40)의 개략적인 블록 다이어그램이다. 호스트 프로세서(42)에 추가하여, 상기 피어-벡터 머신(40)에는 적어도 일부의 데이터 처리를 수행하여 도 1의 컴퓨팅 머신(10) 내의 코프로세서(14)의 뱅크(bank)를 유효하게 대체하는 파이프라인 가속기(44)가 포함되어 있다. 따라서, 호스트-프로세서(42) 및 가속기(44)는 데이터 벡터를 앞뒤로 전송할 수 있는 "피어(peer)" 이다. 가속기(44)는 프로그램 명령을 실행하지 않으므로, 가속기는 종종 코프로세서의 뱅크가 주어진 클록 주파수에서 할 수 있는 것보다 훨씬 빠르게 데이터상의 집중적인 연산을 수학적으로 처리한다. 따라서, 프로세서(42)의 결정-형성 가능성과 가속기(44)의 수-크런칭 가능성을 결합함으로써, 머신(40)은 동일한 능력을 갖지만, 머신(10)과 같은 종래의 컴퓨팅 머신보다는 빠르게 데이터를 처리할 수 있다. 또한, 앞서 언급한 발명의 명칭이 "PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD" 인 미국 특허출원 제10/683,929호에 설명된 바와 같이, 가속기(44)에 상기 호스트 프로세서(42)가 머신(40)의 설계 및 수정을 용이하게 해 주는 것과 동일한 통신 인터페이스를 제공하는데, 특히, 상기 통신 인터페이스가 산업 표준인 경우 그러하다. 가속기(44)에 다수의 구성요소(예를 들어, PLIC)가 포함되어 있고, 이들 구성요소에 이 동일한 인터페이스를 제공하는 것은 가속기의 설계 및 수정을 용이하게 해주는데, 특히 상기 통신 인터페이스가 산업-표준인 경우 그러하다. 더욱이, 머신(40)은 후술하는 바와 같은 그리고 앞서 언급한 다른 출원들에서의 다른 장점도 제공한다.

도 3을 계속 참조하면, 호스트 프로세서(42) 및 파이프라인 가속기(44)에 추가하여, 피어-벡터 컴퓨팅 머신(40)에는 프로세서 메모리(46), 인터페이스 메모리(48), 버스(50), 펌웨어 메모리(52), 선택적인 원-데이터 입력 포트(54 및 56), 처리된-데이터 출력 포트(58 및 60), 및 선택적인 라우터(61)가 포함되어 있다.

호스트 프로세서(42)에는 처리 유닛(62) 및 메시지 처리기(64)가 포함되어 있으며, 프로세서 메모리(46)에는 처리-유닛 메모리(66) 및 처리기 메모리(68)을 포함하는데, 각각 프로세서 유닛 및 메시지 처리기를 위한 프로그램 및 작업 메모리 모두의 역할을 한다. 프로세서 메모리(46)에는 가속기-구성 레지스트리(70) 및 메시지-구성 레지스트리(72)도 포함되어 있는데, 이들은 각각 호스트 프로세서(42)로 하여금 가속기(44)의 기능 및 메시지 처리기(64)가 송수신하는 메시지의 구조를 구성하도록 하는 각각의 구성 데이터를 저장하고 있다.

파이프라인 가속기(44)는 적어도 하나의 PLIC(도시하지 않음)에 배치되어 있으며 프로그램 명령을 실행하지 않고 각각의 데이터를 처리하는 하드와이어드 파이프라인(74_1-74_n)을 포함하고 있다. 펌웨어 메모리(52)는 가속기(44)용 구성 펌웨어를 저장한다. 만일 가속기(44)가 다수의 회로 보드상에 배치된다면, 이들 PLIC와 그의 각각의 펌웨어 메모리는 더터 카드(daughter card)(도시하지 않음)와 같은 회로 보드상에 배치된다. 상기 가속기(44)과 더터 카드는 앞서 언급한 발명의 명칭이 "PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD"인 미국 특허출원 제10/683,929호, 및 발명의 명칭이 "PIPELINE ACCELERATOR HAVING MULTIPLE PIPELINE UNITS AND RELATED COMPUTING MACHINE AND METHOD"인 미국 특허출원 제10/683,932호에 설명되어 있다. 선택적으로, 상기 가속기(44)는 적어도 하나의 ASIC 에 배치될 수 있으며, 따라서, 구성할 수

없는 내부 상호접속을 갖는다. 이 대안에서, 머신(40)에서 펌웨어 메모리(52)를 생략할 수 있다. 또한, 비록 가속기(44)가 다중 파이프라인(74)을 포함하는 것으로 도시되어 있으나, 오직 하나의 파이프라인만 포함할 수 있다. 또한, 비록 도시하지는 않았지만, 가속기(44)에는 디지털-신호 처리기(DSP)와 같은 하나 또는 그 이상의 프로세서가 포함될 수 있다.

피어-벡터 머신(40)의 일반적인 동작은 앞서 언급한 발명의 명칭이 "IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD"인 미국 특허출원 제10/684,102호에 설명되어 있고, 호스트 프로세서(42)의 토폴로지 및 동작은 도 4 내지 도 7을 참조하여 후술한다.

도 4는 본 발명의 일 실시예에 따른 도 3의 호스트 프로세서(42) 및 상기 파이프라인 버스(50)의 기능 블럭 다이어그램이다. 일반적으로, 처리 유닛(62)은 하나 또는 그 이상의 소프트웨어 애플리케이션을 실행하고 메시지 처리기(64)는 소프트웨어 애플리케이션과 파이프라인 가속기(44)(도 3)간의 데이터를 전송하는 하나 또는 그 이상의 소프트웨어 오브젝트(object)를 수행한다. 데이터-처리 스플리팅, 데이터-전송, 및 서로 다른 애플리케이션과 오브젝트들 간의 다른 기능들은 설계자에게 호스트-프로세서 소프트웨어의 설계 및 수정을 보다 쉽게 해준다. 또한, 이하 설명에서 소프트웨어 애플리케이션을 특정 연산을 수행하는 것으로 설명하였으나, 실제 연산에서, 처리 유닛(62) 또는 메시지 처리기(64)가 상기 애플리케이션 소프트웨어를 실행하고 상기 애플리케이션의 제어하에 이 연산을 수행하는 것으로 이해된다. 비슷하게, 비록 이래 설명에서는 소프트웨어 오브젝트를 특정 연산을 수행하는 것으로 설명하였으나, 실제 연산에서, 상기 처리 유닛(62) 또는 메시지 처리기(64)가 상기 소프트웨어 오브젝트를 실행하고 상기 오브젝트의 제어하에 이 연산을 수행하는 것으로 이해된다.

도 4를 계속 참조하면, 처리 유닛(62)은 데이터-처리 애플리케이션(80), 가속기 에러 관리자 애플리케이션(이하 에러 관리자)(82), 및 가속기 구성 관리자 애플리케이션(이하 구성 관리자)(84)을 실행하는데, 이들을 총칭하여 처리-유닛 애플리케이션으로 부른다. 상기 데이터-처리 애플리케이션은 파이프라인 가속기(44)(도 3)의 협력하여 데이터를 처리한다. 예를 들어, 데이터-처리 애플리케이션(80)은 포트(54)(도 3)를 통해 원 소나 데이터를 수신하고, 상기 데이터를 구문분석(parse)하고, 그리고 상기 구문분석된 데이터를 가속기(44)로 전송하며, 상기 가속기는 상기 구문분석된 데이터상에서 FFT를 수행하고 처리된 데이터를 상기 데이터-처리 애플리케이션으로 추가 처리를 위해 돌려보낸다. 상기 에러 관리자(82)는 가속기(44)로부터 에러 메시지를 처리하고, 상기 구성 관리자(84)는 상기 피어-벡터 머신(40)(도 3)의 초기화가 진행되는 동안 상기 가속기의 구성 펌웨어를 상기 메모리(52)로 로드한다. 상기 구성 관리자(84)는 초기화 후에 상기 가속기의 고장 등에 응답하여 가속기(44)를 재구성하기도 한다. 도 6-7을 참조하여 후술하는 바와 같이, 상기 처리-유닛 애플리케이션은 경선(85,87,89)으로 표시된 바와 같이 서로간에 직접 통신을 하거나 또는 데이터-전송 오브젝트(86)를 통해 서로간에 통신을 한다. 메시지 처리기(64)는 상기 데이터-전송 오브젝트(86), 통신 오브젝트(88), 및 입출력 완독 오브젝트(90,92)를 실행하고, 입출력 큐 오브젝트(94,96)를 실행한다. 상기 데이터-전송 오브젝트(86)는 상기 통신 오브젝트(88) 및 상기 처리-유닛 애플리케이션간의 데이터를 전송하고, 상기 처리-유닛 애플리케이션과 상기 가속기(44)가 독립적으로 동작하도록 하게 하는 데이터로서 상기 인터페이스 메모리(48)를 사용한다. 예를 들어, 메모리(48)는 상기 데이터-처리 애플리케이션(80)보다 종종 더 빠른 가속기(44)로 하여금 데이터-처리 애플리케이션을 위해 "대기" 하지않고 동작하도록 한다. 통신 오브젝트(88)는 상기 데이터 오브젝트(86)와 상기 파이프라인 버스(50) 사이에 데이터를 전송한다. 상기 입출력 완독 오브젝트(90,92)는 상기 통신 오브젝트(88)와 상기 처리-유닛 애플리케이션 사이의 데이터를 전송하는 것처럼 데이터-전송 오브젝트(86)를 제어한다. 그리고, 실행되면, 입출력 큐 오브젝트(94,96)는 상기 입출력 완독 오브젝트(90,92)가 원하는 우선순위에 따라 데이터의 어려한 전송을 동기화하게 한다.

또한, 피어-벡터 머신(40)(도 3)의 초기화 동안에, 메시지 처리기(64)는 상기 메시지-구성 레지스트리(72)(도 3)에 저장된 구성 데이터로부터 상기 데이터-전송 오브젝트(86)를 인스턴스생성(instantiate)하는 공개의 오브젝트 팩토리(object factory)(98)를 인스턴스생성하고 실행한다. 상기 메시지 처리기(64)는, 상기 메시지 구성 레지스트리(72)에 저장되어 있는 구성 데이터로부터 상기 통신 오브젝트(88), 상기 입출력 완독기 오브젝트(90,92), 및 상기 입출력 큐 오브젝트(94,96)를 인스턴스생성 하기도 한다. 따라서, 설계자는 상기 레지스트리(72)내에 저장된 구성 데이터를 단지 설계 및 수정하는 것만으로 이들 소프트웨어 오브젝트 및 이들의 데이터-전송 파라메터들을 설계 및 수정할 수 있다.

도 4의 상기 호스트 프로세서(42)의 동작을 도 6-7을 참조하여 아래에 설명한다.

데이터 처리

도 5는, 본 발명의 일 실시예에 따른 도 4의 데이터-처리 애플리케이션(80), 데이터-전송 오브젝트(86), 및 인터페이스 메모리(48)의 기능 블럭 다이어그램이다.

상기 데이터-처리 애플리케이션(80)에는, 각각의 데이터-처리 연산을 각각 수행하는 다수의 스레드(thread)(100_1-100_n)가 포함되어 있다. 예를 들어, 스레드(100_1)는 덧셈을 수행하고, 스레드(100_2)는 뺄셈을 수행하고, 또는 두 스레드(100_3,100_4) 모두 덧셈을 수행해도 좋다.

스레드(100) 각각은 파이프라인 가속기(44)(도 3)를 위한 데이터 발행(publish)을 생성하고, 상기 가속기로부터의 데이터 가입(subscribe)을 수신하고, 또는 데이터로 발행과 가입을 모두 한다. 예를 들어, 스레드(100_1-100_4) 각각은 가속기(44)로부터 데이터를 발행 및 가입한다. 스레드(100)는 또한 다른 스레드(100)와 직접 통신하기도 한다. 예를 들어, 점선(102)으로 표기한 바와 같이, 스레드(100_5-100_4)는 서로 직접 통신한다. 또한, 스레드(100)는 상기 가속기(44)(도 3) 이외의 구성요소(도시하지 않음)로/부터 데이터를 수신 및 전송한다. 간략화를 위해, 그러한 구성요소와 스레드(100)간의 데이터 전송에 대한 설명은 생략한다.

도 5를 계속 참조하면, 인터페이스 메모리(48) 및 데이터-전송 오브젝트(86_{1a}-86_{nb})는 개개의 스레드(100) 및 통신 오브젝트(88)간의 데이터 전송을 위한 다수개의 단방향 채널(104_1-104_n)을 기능적으로 형성한다. 인터페이스 메모리(48)에는 채널당 하나의 버퍼인 다수의 버퍼(106_1-106_n)가 포함된다. 상기 버퍼(106) 각각은 데이터의 단일 그루핑(예를 들어, 바이트, 워드, 블록)을 보유하고 있고, 또는 상기 버퍼의 적어도 일부는 각각이 개별적인 데이터의 다중 그루핑을 저장할 수 있는 FIFO 버퍼일 수 있다. 또한, 채널(104)당 두 개의 데이터 오브젝트(86)도 있을 수 있는데, 하나는 각각의 스레드(100)와 각각의 버퍼(106) 사이의 데이터를 전송하고, 다른 하나는 상기 버퍼(106)와 통신 오브젝트(88) 사이의 데이터 전송을 한다. 예를 들어, 채널(104_1)에는 버퍼(106_1), 스레드(100_1)로부터 버퍼(106_1)까지의 상기 발행된 데이터를 전송하기 위한 데이터-전송 오브젝트(86_{1a}), 및 버퍼(106_1)로부터 통신 오브젝트(88)까지 상기 발행된 데이터를 전송하기 위한 데이터-전송 오브젝트(86_{1b})가 포함된다. 각각의 허용가능한 데이터를 위한 개개의 채널(104)을 포함하는 것은 잠재적인 데이터 병목을 감소시키고 호스트 프로세서(42)(도 4)의 설계 및 수정을 용이하게 하기도 한다.

도 3 내지 도 5를 참조하면, 초기화 동안 및 데이터-처리 애플리케이션(80), 데이터-전송 오브젝트(86), 통신 오브젝트(88) 및 선택적인 확독기 및 뷰 오브젝트(90,92,94,96)가 실행되는 동안의 호스트 프로세서(42)의 동작을 본 발명의 일 실시예에 따라 설명한다.

호스트 프로세서(42)의 초기화 동안, 오브젝트 팩토리(98)는 데이터-전송 오브젝트(86)를 인스턴스생성하고 상기 버퍼(104)를 정의한다. 특히, 오브젝트 팩토리(98)은 상기 레지스트리(72)로부터 구성 데이터를 다운로드하고 상기 데이터-처리 애플리케이션(80)이 필요로 하는 각각의 데이터-전송 오브젝트(86_{xb})를 위한 소프트웨어 코드를 생성한다. 상기 애플리케이션(80)이 필요로 하는 데이터-전송 오브젝트(86_{xb})의 식별은 보통 구성 데이터의 일부이지만, 애플리케이션(80)은 데이터-전송 오브젝트(87) 모두를 사용할 필요는 없다. 따라서, 생성된 오브젝트(86_{xb})로부터, 오브젝트 팩토리(98)는 각각 데이터 오브젝트(86_{xa})를 인스턴스생성한다. 일반적으로, 아래의 실시예에서 설명한 바와 같이, 오브젝트 팩토리(98)는 동일한 버퍼(104)를 동일한 소프트웨어 코드의 다중 인스턴스로서 액세스하는 데이터-전송 오브젝트(86_{xa}, 86_{xb})를 인스턴스생성 한다. 이것은 오브젝트 팩토리(98)가 생성하는 코드의 양을 거의 절반 감소시킨다. 또한, 메시지 처리기(64)는 데이터-전송 오브젝트(86)가 애플리케이션(80)을 필요로 하지 않은 경우를 결정하고 이를 필요로 하지 않는 데이터-전송 오브젝트의 인스턴스를 삭제해서 메모리를 절약한다. 선택적으로, 메시지 처리기(64)는, 오브젝트 팩토리(98)가 데이터-전송 오브젝트(86)을 생성하기 전에 이 결정을 해서, 오브젝트 팩토리가 애플리케이션(80)이 필요로 하는 데이터-전송 오브젝트만을 인스턴스생성 하도록 한다. 또한, 데이터-전송 오브젝트(86)에는 각각의 버퍼(104)가 위치하는 인터페이스 메모리(48)의 어드레스가 포함되어 있기 때문에, 오브젝트 팩토리(98)는 데이터-전송 오브젝트를 인스턴스생성하는 경우 상기 버퍼의 크기와 위치를 효과적으로 정의한다.

예를 들어, 오브젝트 팩토리(98)는 아래와 같은 방법으로 데이터-전송 오브젝트(86_{1a}-86_{1b})를 인스턴스생성 한다. 먼저, 팩토리(98)는 레지스트리(72)로부터 구성 데이터를 다운로드하고, 데이터-전송 오브젝트(86_{1a}-86_{1b})를 위한 공통 소프트웨어를 생성한다. 다음으로, 팩토리(98)는 상기 공통 소프트웨어 코드의 각각의 인스턴스로서 데이터-전송 오브젝트(86_{1a},86_{1b})를 인스턴스생성 한다. 즉, 메시지 처리기(64)는 상기 공통 소프트웨어 코드를 상기 처리기 메모리(86)의 두 위치로 또는 다른 프로그램 메모리(도시하지 않음)로 효과적으로 복사하고, 오브젝트(86_{1a})에 대한 한 위치 및 오브젝트(86_{1b})를 위한 다른 위치를 실행한다.

도 3 내지 도 5를 계속 참고하면, 호스트 프로세서(42)의 초기화 후에, 데이터-터리 애플리케이션(80)가 데이터를 처리하고, 파이프라인 가속기(44)로/로부터 데이터를 전송 및 수신한다.

상기 가속기(44)로 데이터를 전송하는 데이터-처리 애플리케이션(80)의 한 예를 채널(104_1)를 참고하여 설명한다.

먼저, 스레드(100_1)가 상기 데이터-전송 오브젝트(86_{1a})로 데이터를 생성 및 발행한다. 상기 스레드(100_1)는, 상기 가속기(44)로부터 또는 소나 어레이와 같은 다른 소스(도시하지 않음)로부터 수신되는 원 데이터 또는 프로토(64)를 통한 데이터 베이스상에서의 연산에 의해 상기 데이터를 생성하기도 한다(후술함).

이어서, 데이터-오브젝트(86_{1a})는 상기 발행된 데이터를 버퍼(106_1)로 로드한다.

다음으로, 데이터-전송 오브젝트(86_{1b})는 버퍼(106_1)에 상기 데이터-전송 오브젝트(86_{1a})로부터 새롭게 발행된 데이터가 로드되었는지를 결정한다. 출력 완독기 오브젝트(92)는 상기 데이터-전송 오브젝트(86_{1b})에게 주기적으로 명령하여 새롭게 발행된 데이터를 위한 버퍼(106_1)를 검사하게 한다. 선택적으로, 출력 완독기 오브젝트(92)는, 버퍼(106_1)가 새로이 발행된 데이터를 수신하면 상기 데이터-전송 오브젝트(86_{1b})에게 통보한다. 특히, 출력 큐 오브젝트(96)는 상기 버퍼(106_1)내에 상기 발행된 데이터를 저장하는 데이터-전송 오브젝트(86_{1a})에 응답하여 고유 식별기(도시하지 않음)를 생성 및 저장한다. 이 식별기에 응답하여, 출력 완독기 오브젝트(92)는 상기 버퍼(106_1)가 새롭게 발행된 데이터를 포함하고 있음을 상기 데이터-전송 오브젝트(86_{1b})에게 통보한다. 다수의 버퍼(106)들에는 각각 새롭게 발행된 데이터가 포함되어 있어서, 출력 큐 오브젝트(96)는 이 데이터가 발행된 순서를 기록하고, 출력 완독기 오브젝트(92)는 상기 각각의 데이터-전송 오브젝트(86_{1b})에게 이 순서를 통보한다. 따라서, 출력 완독기 오브젝트(92)와 출력 큐 오브젝트(96)는, 상기 발행된 제1 데이터가 각각의 데이터-전송 오브젝트(86_{2b})가 가속기(44)에게 전송하는 제1 데이터가 되도록 하고, 상기 발행된 제2 데이터가 상기 각각의 데이터-전송 오브젝트(86_{2b})가 상기 가속기로 전송하는 제2 데이터가 되도록 하는 것을 통해 데이터 전송을 동기화 한다. 다른 대안으로서, 다수의 버퍼(106)에 각각 새롭게 발행된 데이터가 포함되어 있으므로, 출력 완독기 및 출력 큐 오브젝트(92,96)는 선입선출 스킴(scheme)과는 다른 또는 그것에 추가하여 우선순위 스킴을 수행한다. 예를 들어, 스레드(100_1)가 제1 데이터를 발행한다고 가정하면, 이어서 스레드(100_2)가 제2 데이터를 발행하고 상기 출력 큐 오브젝트(96)로 상기 제2 데이터와 관련된 우선순위 프래그도 발행한다. 상기 제2 데이터는 상기 제1 데이터보다 우선순위를 가지고 있기 때문에, 출력 완독기 오브젝트(92)는, 상기 버퍼(106_1)내의 상기 발행된 제1 데이터의 데이터-전송 오브젝트(86_{1b})를 통보하기 전에, 상기 버퍼(106_2)내의 상기 발행된 제2 데이터의 데이터-전송 오브젝트(86_{2b})를 통보한다.

다음으로, 데이터-전송 오브젝트(86_{1b})는 상기 버퍼(106_1)로부터 상기 발행된 데이터를 검색하고, 미리 결정된 방식으로 상기 데이터를 포맷한다. 예를 들어, 상기 오브젝트(86_{1b})는 상기 발행된 데이터를 포함하는 메시지(예를 들어, 페이로드) 및 가속기(44) 내부의 상기 데이터의 목적지를 나타내는 헤더를 생성한다. 이 메시지는 Rapid IO(입력/출력) 포맷과 같은 산업-표준 포맷을 가질 수 있다.

데이터-전송 오브젝트(86_{1b})는 발행된 데이터를 포맷한 후, 상기 통신 오브젝트(88)로 이 포맷된 데이터를 전송한다.

다음으로, 통신 오브젝트(88)가 상기 포맷된 데이터를 버스(50)를 통해 파이프라인 가속기(44)로 전송한다. 통신 오브젝트(88)는 호스트 프로세서(42)와 가속기(44)간의 데이터 전송에 사용되는 통신 프로토콜(예를 들어, Rapid IO, TCP/IP)를 수행하도록 설계되었다. 예를 들어, 통신 오브젝트(88)는 상기 프로토콜이 필요로 하는 요구되는 핸드 세이킹(hand shaking) 및 다른 전송 파라메터들(예를 들어, 상기 버스(50)상의 메시지 송수신을 조정하는)을 수행한다. 선택적으로, 데이터-전송 오브젝트(86_{1b})가 상기 통신 프로토콜을 수행할 수 있고, 통신 오브젝트(88)는 생략할 수 있다. 그러나, 후자의 경우는 추가적인 코드와 기능을 포함하도록 모든 데이터-전송 오브젝트(86_{1b})를 요구하기 때문에 효율은 낮다.

다음으로, 파이프라인 가속기(44)가 상기 포맷된 데이터를 수신하고, 상기 메시지로부터 이 데이터를 복구(예를 들어, 헤더가 있다면 이 헤더에서 상기 데이터를 분리함)하고, 상기 데이터를 가속기 내부의 적절한 목적지로 향하게 한다.

도 3 내지 도 5를 계속 참조하면, 호스트 프로세서(42)(도 3)로 데이터를 전송하는 파이프라인 가속기(44)(도 3)의 실시예를 채널(104_2)를 참조하여 설명한다.

먼저, 파이프라인 가속기(44)가 데이터를 생성하고 포맷한다. 예를 들어, 가속기(44)는 상기 데이터 페이로드 및 상기 데이터를 수신하고 처리하는 스레드인 목적지 스레드(100_1,100_2)를 식별하는 헤더를 포함하는 메시지를 생성한다. 상기 설명한 바와 같이, 이 메시지는 Rapid IO(입력/출력) 포맷과 같은 산업 표준 포맷을 가지고 있다.

다음으로, 가속기(44)가 종래 방식으로 상기 버스(50)상에서 상기 포맷된 데이터를 구동시킨다.

이어서, 통신 오브젝트(88)가 상기 버스(50)로부터 상기 포맷된 데이터를 수신하고, 데이터-전송 오브젝트(86_{2b})에게 상기 포맷된 데이터를 제공한다. 한 실시예에서, 상기 포맷된 데이터는 메시지 형태이고, 통신 오브젝트(88)가 상기 메시지 헤더를 분석하고(상기 설명한 바와 같이, 목적지 스레드(100_1,100_2)를 식별함), 상기 헤더에 응답하여 데이터-전송 오브젝트(86_{2b})에게 상기 메시지를 제공한다. 다른 실시예에서, 상기 통신 오브젝트(88)가 상기 데이터-전송 오브젝트(86_{2b})로부터 상기 메시지를 제공하는데, 상기 오브젝트 각각은 상기 목적지 스레드(100_1,100_2)로 데이터를 제공하도록 기능이 있는 경우에만 상기 메시지를 처리하고 상기 메시지를 분석한다. 따라서, 이 실시예에서, 오직 데이터-전송 오브젝트(86_{2b})만이 상기 메시지를 처리한다.

다음으로, 상기 데이터-전송 오브젝트(86_{2b})가 상기 통신 오브젝트(88)로부터 수신된 데이터를 버퍼(106_2)로 로드한다. 예를 들어, 만일 데이터가 메시지 페이로드 내부에 포함되어 있다면, 상기 데이터-전송 오브젝트(86_{2b})는 상기 메시지로부터 상기 데이터를 복구(예를 들어, 상기 헤더를 스트립(strip)함)하고 이 복구된 데이터를 상기 버퍼(106_2)로 로드한다.

이어서, 상기 데이터-전송 오브젝트(86_{2a})는 상기 버퍼(106_2)가 상기 데이터-전송 오브젝트(86_{2b})로부터 수신된 새로운 데이터를 가지는지를 결정한다. 입력 완독기 오브젝트(90)는 데이터-전송 오브젝트(86_{2a})에게 주기적으로 명령하여 새롭게 수신된 데이터가 있는지 상기 버퍼(106_2)를 점사하게 한다. 선택적으로, 입력 완독기 오브젝트(90)는, 상기 버퍼(106_2)에 새롭게 발행된 데이터가 수신되면 상기 데이터-전송 오브젝트(86_{2a})에게 통보한다. 특히, 입력 큐 오브젝트(94)가 상기 버퍼(106_2)내의 상기 발행된 데이터를 저장하는 상기 데이터-전송 오브젝트(86_{2a})에 응답하여 고유 식별기(도시하지 않음)를 생성하고 저장한다. 이 식별기에 응답하여, 입력 완독기 오브젝트(90)는, 버퍼(106_2)가 새로이 발행된 데이터를 포함하는 것을 상기 데이터-전송 오브젝트(86_{2a})에게 통보한다. 출력 완독기 및 출력 큐 오브젝트(92,96)를 참고하여 앞서 설명한 바와 같이, 다수의 버퍼(106)가 각각 새롭게 발행된 데이터를 포함하고 있으면, 입력 큐 오브젝트(94)는 이 데이터가 발행된 순서대로 기록을 하고, 입력 완독기 오브젝트(90)가 그 순서대로 상기 각각의 데이터-전송 오브젝트(86_{2a})로 통보한다. 선택적으로, 다수의 버퍼(106)에 각각 새롭게 발행된 데이터가 포함되어 있으면, 상기 입력 완독기 및 입력 큐 오브젝트(94,96)는 선입선출 스킴과는 다른 또는 그에 추가하여 우선순위 스킴을 수행한다.

다음으로, 데이터-오브젝트(86_{2a})는 이 데이터를 버퍼(106_2)로부터 상기 데이터상에 각각의 연산을 수행하는 상기 서브 스프라이브 스레드(100_1,100_2)로 전송한다.

도 5를 참고하면, 다른 스레드로부터 데이터를 수신하고 처리하는 한 스레드의 예를 상기 스레드(100_3)가 발행한 데이터를 수신하고 처리하는 스레드(100_4)를 참고하여 설명한다.

한 실시예에서, 스레드(100_3)는 선택적인 접속(점선)(102)을 통해 상기 스레드(100_4)로 직접 데이터를 발행한다.

다른 실시예에서, 상기 스레드(100_3)는 채널(104_5,104_6)을 통해서 상기 스레드(100_4)로 데이터를 발행한다. 특히, 상기 데이터-전송 오브젝트(86_{3a})는 상기 발행된 데이터를 버퍼(106_3)로 로드한다. 다음으로, 데이터-전송 오브젝트(86_{3b})가 상기 버퍼(106_3)로부터 상기 데이터를 검색하고, 이 데이터를 데이터-전송 오브젝트(86_{4b})로 발행하는 통신 오브젝트(88)로 이 데이터를 전송한다. 그 다음, 데이터-전송 오브젝트(86_{4b})는 이 데이터를 버퍼(106_4)에 로드한다. 다음으로, 데이터

-전송 오브젝트(86_{5b})는 이 데이터를 상기 버퍼(106_6)로부터 상기 스레드(100_4)로 전송한다. 선택적으로, 이 데이터는 버스(50)를 통해 전송되지 않기 때문에, 설계자는 상기 데이터를 버퍼(106_6)에 직접 로드하여 통신 오브젝트(88)와 데이터-전송 오브젝트(86_{5b})를 바이패스하도록 상기 데이터-전송 오브젝트(86_{5b})를 수정할 수 있다. 그러나, 상기 데이터-전송 오브젝트(86_{5b})를 다른 데이터-전송 오브젝트(86)와 다르도록 수정하는 것은 메시지 처리기(64)의 모듈성(modularity)의 복잡성을 증가시킨다.

도 6을 계속 참고하면, 추가적인 데이터-전송 기술이 수행된다. 예를 들어, 하나의 스레드가 각각의 다중 채널을 통해 파이프라인 가속기(44)(도 3) 내부의 여러 위치로 데이터를 발행하기도 한다. 선택적으로, 앞서 언급한 발명의 명칭이 "IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD" 인 미국 특허출원 제10/684,102호, 및 발명의 명칭이 "PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD" 인 미국 특허출원 제10/683,929호에 설명된 바와 같이, 가속기(44)는 하나의 채널(104)을 통해 데이터를 수신하고 이것을 상기 가속기 내부의 여러 위치에 제공한다. 또한, 여러 스레드(예를 들어, 100_1 및 100_2)는 동일한 채널(예를 들어, 104_2)로부터 데이터를 서브스크라이브 한다. 또한, 여러 스레드(예를 들어, 100_3 및 100_5)는, 비록 이들 스레드가 각각의 채널(104)을 통해 동일한 가속기 위치로 데이터를 발행하긴 하지만, 동일한 채널(예를 들어, 104_3)을 통해 상기 가속기(44) 내부의 동일한 위치로 데이터를 발행한다.

도 6은 본 발명의 한 실시예에 따른 상기 예외 관리자(82), 데이터-전송 오브젝트(86), 및 인터페이스 메모리(48)의 기능블럭 다이어그램이다.

예외 관리자(82)는 파이프라인 가속기(44)(도 3)의 초기화 또는 동작 동안에 발생하기도 하는 예외들을 수신하고 로그(log) 한다. 일반적으로, 가속기(44)가 원치 않는 방식으로 작동하는 것과 같은 설계자가 정의한 이벤트이다. 예를 들어, 오버플로우 되는 버퍼(도시하지 않음)가 상기 예외 일 수 있어서, 가속기(44)가 예외 메시지를 생성하고 이것을 예외 관리자(82)에게 전송하게 한다. 예외 메시지의 생성은 앞서 언급한 발명의 명칭이 "PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD" 인 미국 특허출원 제10/683,929호에 설명되어 있다.

상기 예외 관리자(82)는 파이프라인 가속기(44)(도 3)의 초기화 또는 동작 중에 발생하는 예외를 처리하기도 한다. 예를 들어, 만일 가속기(44)에 오버플우우하는 버퍼(도시하지 않음)가 포함되어 있다면, 예외 관리자(82)는 상기 가속기로 하여금 장래의 오버플로우를 예방하도록 상기 버퍼의 크기를 증가시키도록 한다. 또는, 가속기(44)의 일부가 고장나면, 상기 예외 관리자(82)는 가속기 또는 데이터-처리 애플리케이션(80)의 다른 부분으로 하여금 상기 고장 부분이 수행하도록 의도된 연산을 수행하게 한다. 이러한 예외 처리는 앞서 언급한 발명의 명칭이 "PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD" 인 미국 특허출원 제10/683,929호에 더 설명되어 있다.

가속기 예외를 로드 및/또는 처리하기 위해서는, 상기 예외 관리자(82)는 하나 또는 그 이상의 서브스크라이버 스레드(100)(도 5)로부터 데이터를 서브스크라이브 하고 이 데이터로부터 예외가 발생했는지를 결정한다.

한 대안으로서, 상기 예외 관리자(82)는 상기 서브스크라이버 스레드(100)(도 5)가 서브스크라이브한 것과 동일한 데이터를 서브스크라이브한다. 특히, 관리자(82)는, 상기 서브스크라이버 스레드(100)(도 5의 스레드(100_1 및 100_2)를 포함)가 데이터를 수신하는 동일한 각각의 채널(104_3)(도 5의 채널(104_2))을 통해 이 데이터를 수신한다. 따라서, 채널(104_2)은, 이 채널들이 상기 데이터를 서브스크라이버 스레드(100)로 제공하는 것과 동일한 방식으로, 이 데이터를 예외 관리자(82)로 제공한다.

다른 대안에서, 예외 관리자(82)는, 서브스크라이버 채널(104_3)을 통해 데이터를 스레드(100)로 제공하지 않는 가속기(44)(도 3)의 부분으로부터 데이터를 수신하기도 하는 전용 채널(106)(도시하지 않음)로부터 데이터를 서브스크라이브 한다. 그러한 전용 채널(106)을 사용하여, 오브젝트 팩토리(98)(도 4)는, 도 4를 참고하여 앞서 설명한 바와 같이 호스트 프로세서(42)의 초기화 동안에, 이들 채널을 위한 데이터-전송 오브젝트(86)를 생성한다. 상기 예외 관리자(82)는 상기 서브스크라이버 채널(104_3)에 더하여 또는 배타적으로 전용 채널(106)로 서브스크라이브 한다.

예외가 발생하는지를 결정하기 위해서, 예외 관리자(82)는 이 데이터를 메모리(66) 내부의 레지스트리(도시하지 않음)내에 저장된 예외 코드와 비교한다. 만일, 상기 데이터가 코드의 하나와 매치하면, 예외 관리자(82)는 상기 매치된 코드에 대응하는 예외가 발생하였음을 결정한다.

더 다른 대안에서, 상기 예외 관리자(82)는 이 데이터를 분석하여 예외가 발생했는지를 결정한다. 예를 들어, 상기 데이터는 가속기(44)에 의해 수행된 연산의 결과를 나타내기도 한다. 상기 예외 관리자(82)는 이 데이터가 에러를 포함하는지를 결정하고, 만약 포함한다면, 예외가 발생했고 그 예외의 식별을 결정한다.

예외가 발생했음을 결정한 후, 예외 관리자(82)는, 나중에 가속기(44)의 디버그 동안에 이를 사용하기 위해서, 예를 들어 대응하는 예외 코드 및 발생 시간을 로그한다. 예외 관리자(82)는 상기 예외의 식별을 결정하고, 예를 들어, 이를 종래 방식에서의 시스템 설계자에게 전달하기도 한다.

선택적으로, 상기 예외를 로그하는데 추가하여, 예외 관리자(82)는 상기 예외를 처리하기 위한 적절한 절차를 수행한다. 예를 들어, 예외 관리자(82)는 가속기(44), 데이터-처리 애플리케이션(80) 또는 구성 관리자(84)에게 예외-처리 명령을 보내어 상기 예외를 처리한다. 상기 예외 관리자(82)는, 상기 발행기 스레드(100)(예를 들어, 도 5의 100_1)가 데이터를 발행하는 동일한 각각의 채널(104_p)(예를 들어, 도 5의 채널(104_1))을 통해, 또는 도 5를 참고하여 상기 설명한 바와 같은 동작을 하는 전용 예외-처리 채널(104)(도시하지 않음)을 통해 상기 예외-처리 명령을 상기 가속기(44)로 보낸다. 만일 예외 관리자(82)가 다른 채널(104)을 통해 명령을 보내면, 오브젝트 팩토리(98)(도 4)는 도 4를 참고하여 상기 설명한 바와 같이 호스트 프로세서(42)의 초기화 동안 이들 채널을 위한 데이터-전송 오브젝트(86)을 생성한다. 예외 관리자(82)는 상기 데이터-처리 애플리케이션(80) 및 구성 관리자(84)에게 핸들(도 4에서 참신(85,89)으로 표기) 또는 채널(104_{dma1},104_{dma2})(애플리케이션(80)) 및 채널(104_{cm1},104_{cm2})(구성 관리자(84)),을 통해 상기 예외-처리 명령을 발행하는데, 상기 채널들은 상기 오브젝트 팩토리(98)가 호스트 프로세서(42)의 초기화 동안 생성하기도 하는 것이다.

도 6을 계속 참고하면, 후술하는 바와 같이, 상기 예외-처리 명령은 가속기(44), 데이터-처리 애플리케이션(80), 또는 구성 관리자(84)로 하여금 다양한 방법으로 대응하는 예외를 처리하게 해준다.

가속기(44)로 보내면, 예외-처리 명령은 가속기의 소프트 구성 또는 기능을 변경한다. 예를 들어, 위에서 설명한 바와 같이, 만일 예외가 버퍼 오버플로우라면, 상기 명령은 가속기의 소프트 구성을 버퍼의 크기를 증가시키는 것으로 변경(즉, 소프트 구성 레지스터의 콘텐츠를 변경)한다. 또는, 특정 연산을 수행하는 가속기(44)의 어느 섹션이 고장이라면, 상기 명령은 가속기로 하여금 디스에이블된 섹션을 "오프 라인(off line)"하도록 하여 가속기의 기능을 변경한다. 후자의 경우, 예외 관리자(82)는, 추가의 명령을 통해서, 가속기(44)의 다른 섹션 또는 데이터-처리 애플리케이션(80)으로 하여금 후술하는 바와 같이 디스에이블된 가속기 섹션으로부터 그 연산을 "가져오도록(takr over)" 한다. 가속기(44)의 소프트 구성을 바꾸는 것은 앞서 언급한 발명의 명칭이 "PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD" 인 미국 특허출원 제10/683,929호(대리인 관리번호 1934-13-3)에 더 설명되어 있다.

데이터-처리 애플리케이션(80)으로 보내는 경우에는, 상기 예외-처리 명령은 상기 데이터-처리 애플리케이션으로 하여금 오프 라인되어 있는 가속기(44)의 디스에이블된 섹션의 연산을 "가져오게" 한다. 비록 처리 유닛(62)(도 3)이 이 연산을 가속기(44)보다 느리고 덜 효과적으로 수행하긴 하지만, 이것은 연산을 전혀 수행하지 않는 것에는 적절하다. 가속기(44)로부터 처리 유닛(62)까지 연산의 성능을 시프트할 수 있다는 것은 퍼어-펙티 머신(40)(도 3)의 유연성, 안정성, 지속성, 및 고장에 견디는 힘을 증가시킨다.

구성 관리자(84)로 보내는 경우에는, 예외-처리 명령은 상기 구성 관리자로 하여금 가속기의 하드 구성을 변화하게 하여 가속기가 오프라인으로 유지되어 온 고장 섹션의 연산을 계속 수행할 수 있도록 한다. 예를 들어, 가속기(44)가 미사용된 섹션을 가지고 있다면, 구성 관리자(84)는 이 미사용된 섹션을 구성하여 상기 고장 섹션이였던 연산을 수행하도록 한다. 만약 가속기(44)에 미사용된 섹션이 없다면, 구성 관리자(84)는 제1 연산을 현재 수행하는 가속기의 섹션을 재구성하여 상기 고장 섹션의 제2 연산을 수행, 즉 가져오게 한다. 이 기술은 제1 연산은 생략할 수 있지만 제2 연산을 그렇지 못한 경우, 또는 데이터-처리 애플리케이션(80)이 제2 연산보다는 제1 연산을 수행하기에 보다 적합한 경우에 유용하다. 연산의 성능을 가속기(44)의 어느 한 섹션에서부터 다른 섹션으로 시프트하는 것은 퍼어-펙티 머신(40)(도 3)의 유연성, 안정성, 지속성, 및 고장에 견디는 힘을 증가시킨다.

도 7을 참고하면, 구성 관리자(84)는 상기 피어-벡터 머신(40)(도 3)의 초기화 동안, 가속기(44)의 하드 구성을 정의하는 펌웨어를 로드하고, 또 6을 참고하여 앞서 설명한 바와 같이, 본 발명의 일 실시예에 따른 예외에 응답하여 가속기의 하드 구성을 재정의하는 펌웨어를 로드한다. 아래에 설명한 바와 같이, 구성 관리자(84)는 가속기(44)의 설계와 수정의 복잡성을 줄여주는 하며, 피어-벡터 머신(40)(도 3)의 유연성, 안정성, 지속성, 및 고장에 견디는 힘을 증가시킨다.

피어-벡터 머신(40)의 초기화 동안, 구성 관리자(84)는 가속기 구성 레지스트리(70)로부터 구성 데이터를 수신하고, 상기 구성 데이터에 의해 식별된 구성 펌웨어를 로드한다. 상기 구성 데이터는 상기 펌웨어를 로딩하기 위한 상기 구성 관리자(84)로의 효과적인 명령이다. 예를 들어, 초기화된 가속기(44)의 이노 섹션이 FFT 를 수행한다면, 설계자는 가속기의 이 섹션에서 FFT를 수행하는 가속기(44)에 의해 펌웨어가 로드되도록 상기 구성 데이터를 설계한다. 따라서, 설계자는 피어-벡터 머신(40)의 초기화 이전에 단지 상기 구성 데이터를 생성하거나 수정하는 것만으로 가속기(44)의 하드 구성을 수정할 수 있다. 상기 구성 데이터를 생성하고 수정하는 것이 종종 펌웨어를 직접 생성하고 수정하는 것 보다 쉽기 때문에 - 특히, 구성 데이터는 구성 관리자(84)에게 명령하여 라이브러리로부터 현존 펌웨어를 로드하게 할 수 있다 - 구성 관리자(84)는 일반적으로 가속기(44)의 설계 및 수정의 복잡성을 감소시킨다.

상기 구성 관리자(84)가 구성 데이터에 의해 식별된 펌웨어를 로드하기 전에, 구성 관리자는 가속기(44)가 구성 데이터에 의해 정의된 구성을 지원하는지를 결정한다. 예를 들어, 만약 구성 데이터가 가속기(44)의 특정 PLIC(도시하지 않음)를 위한 펌웨어를 로드하도록 상기 구성 관리자(84)에게 명령하면, 구성 관리자(84)는 상기 PLIC 가 상기 데이터가 로드되기 전에 존재함을 확인한다. 만일 상기 PLIC 가 존재하지 않는다면, 구성 관리자(84)는 가속기(44)의 초기화를 중지시키고 상기 가속기가 상기 구성을 지원하지 않음을 관리자에게 통보한다.

구성 관리자(84)는, 상기 가속기가 상기 정의된 구성을 지원하는 것을 확인한 후, 구성 관리자(84)는 상기 펌웨어를 펌웨어 메모리(62)로 로드함으로써 상기 펌웨어를 가지고 자신의 하드 구성을 설정하는 가속기(44)로 상기 펌웨어를 로드한다. 일반적으로, 구성 관리자(84)는 상기 펌웨어를 생성, 구조 및 동작이 도 5의 채널(104)과 유사한 하나 또는 그 이상의 채널(104_i)을 통해 상기 가속기로 보낸다. 구성 관리자(84)는 하나 또는 그 이상의 채널(104_j)을 통해 가속기(44)로부터 데이터를 수신하기도 한다. 예를 들어, 가속기(44)는 자신의 하드 구성의 성공적인 설정의 확인을 구성 관리자(84)에게 전송한다.

가속기(44)의 하드 구성이 설정된 후, 구성 관리자(84)는 도 6을 참고하여 앞서 설명한 바와 같이 예외 관리자(84)로부터 예외-처리 명령에 응답하여 가속기의 하드 구성을 설정한다. 예외-처리 명령에 응답하여, 구성 관리자(84)는 상기 레지스트리(70)로부터 적절한 구성 데이터를 다운로드하고, 상기 구성 데이터에 의해 식별된 재구성 펌웨어를 로드하고, 이 펌웨어를 채널(104_i)을 통해 가속기(44)로 보낸다. 구성 관리자(84)는 채널(104_j)을 통해 가속기(44)로부터 성공적인 재구성의 확인을 수신한다. 도 6을 참고하여 앞에서 설명한 바와 같이, 구성 관리자(84)는 라인(89)(도 4)를 통해 상기 예외 관리자(84)로부터 직접, 또는 채널(104_{cm1},104_{cm2})을 통해 간접으로 상기 예외-처리 명령을 수신한다.

구성 관리자(84)는 도 6을 참고하여 앞에서 설명한 바와 같이 상기 예외 관리자(84)로부터 예외-처리 명령에 응답하여 상기 데이터-처리 애플리케이션(80)를 재구성하기도 한다. 예외-처리 명령에 응답하여, 구성 관리자(84)는 상기 데이터-처리 애플리케이션(80)에 명령하여 그 자신을 재구성하여 고장 또는 다른 이유로 인해 가속기(44)가 수행할 수 없는 연산을 수행하도록 한다. 구성 관리자(84)는 상기 데이터-처리 애플리케이션(80)에게 라인(87)(도 4)를 통해 직접, 또는, 채널(104_{dp1},104_{dp2})를 통해 간접적으로 명령을 하고, 데이터-처리 애플리케이션으로부터, 성공적인 구성의 확인과 같은 정보를 직접, 또는 다른 채널(104)(도시하지 않음)을 통해 수신한다. 선택적으로, 상기 예외 관리자(82)는 상기 데이터-처리(80)에게, 그 자신을 재구성하는 예외-처리 명령을 보내어 상기 구성 관리자(82)를 바이패스 시킨다.

도 7을 계속 참고하면, 구성 관리자(82)의 다른 실시예가 수행된다. 예를 들어, 구성 관리자(82)는 가속기 고장 발생 이외의 이유를 위해 가속기(44) 또는 데이터-처리 애플리케이션(80)을 재구성한다.

앞의 설명으로부터 당업자는 본 발명을 실시하거나 활용할 수 있다. 본 발명의 실시예들을 당업자는 다양하게 변형시킬 수 있다는 것은 분명하고, 본 발명의 요지와 범위를 벗어나지 않고 다른 실시예 및 응용 분야에 적용할 수 있다. 따라서, 본 발명의 실시예들은 본 발명을 제한하기 위한 것이 아니고 여기에서 공개한 원리와 특징에 넓게 해석되어야 할 것이다.

(57) 청구의 범위

청구항 1.

제1 버퍼; 및

상기 버퍼와 결합되어 있는 프로세서를 구비하고,

상기 프로세서는,

애플리케이션, 제1 데이터-전송 오브젝트, 및 제2 데이터-전송 오브젝트를 실행하고,

상기 애플리케이션의 제어하에 데이터를 발행하고,

상기 제1 데이터-전송 애플리케이션의 제어하에 상기 발행된 데이터를 상기 버퍼에 로드하고,

상기 제2 데이터-전송 오브젝트의 제어하에 상기 버퍼로부터 상기 발행된 데이터를 검색하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 2.

청구항 1에 있어서,

상기 제1 및 제2 데이터-전송 오브젝트는 상기 동일한 오브젝트 코드의 제1 및 제2 인스턴스를 각각 포함하는 것을 특징으로 하는 컴퓨팅 머신.

청구항 3.

청구항 1에 있어서,

상기 프로세서는,

상기 애플리케이션을 실행하고 상기 애플리케이션의 제어하에 상기 데이터를 발행하도록 동작 가능한 처리 유닛; 및

상기 제1 및 제2 데이터-전송 오브젝트를 실행하고, 상기 제1 데이터-전송 오브젝트의 제어하에 상기 발행된 데이터를 상기 버퍼로 로드하고, 그리고 상기 제2 데이터-전송 오브젝트의 제어하에 상기 발행된 데이터를 검색하도록 동작 가능한 데이터-전송 처리기를 포함하는 것을 특징으로 하는 컴퓨팅 머신.

청구항 4.

청구항 1에 있어서,

상기 프로세서는,

상기 애플리케이션의 스레드를 실행하고, 상기 스레드의 제어하에 상기 데이터를 발행하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 5.

청구항 1에 있어서,

상기 프로세서는,

큐 오브젝트 및 완독기 오브젝트를 실행하고,

상기 큐 오브젝트의 제어하에 상기 버퍼로 상기 발행된 데이터의 로딩을 반영하는 값인 큐 값을 저장하고,

상기 완독기 오브젝트의 제어하에 상기 큐 값을 완독하고,

상기 발행된 데이터가 상기 완독기 오브젝트의 제어하에 그리고 상기 큐 값에 응답하여 상기 버퍼를 점유하는 제2 소프트웨어 오브젝트를 통보하고,

상기 제2 데이터-전송 오브젝트의 제어하에 그리고 상기 통보에 응답하여 상기 저장 위치로부터 상기 발행된 데이터를 검색하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 6.

청구항 1에 있어서,

버스를 더 구비하고,

상기 프로세서는, 통신 오브젝트를 실행하고 상기 통신 오브젝트의 제어하에 상기 버스로 상기 검색된 데이터를 구동시키도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 7.

청구항 1에 있어서,

제2 버퍼를 더 구비하고,

상기 프로세서는, 상기 제2 데이터-전송 오브젝트의 제어하에 상기 검색된 데이터를 상기 제2 버퍼로 제공하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 8.

청구항 1에 있어서,

상기 프로세서는, 상기 제2 데이터-전송 오브젝트의 제어하에 헤더 및 상기 검색된 데이터를 조합하는 메시지를 생성하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 9.

청구항 1에 있어서,

상기 제1 및 제2 데이터-전송 오브젝트는 각각 상기 동일한 오브젝트 코드의 제1 및 제2 인스턴스를 포함하고,

상기 프로세서는 오브젝트 팩토리를 실행하고 상기 오브젝트 팩토리의 제어하에 상기 오브젝트 코드를 생성하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 10.

제1 버퍼; 및

상기 제1 버퍼와 결합된 프로세서를 구비하고,

상기 프로세서는,

제1 및 제2 데이터-전송 오브젝트 및 애플리케이션을 실행하고,

데이터를 검색하고 상기 검색된 데이터를 상기 제1 데이터-전송 오브젝트의 제어하에 상기 버퍼로 로드하고,

상기 제2 데이터-전송 오브젝트의 제어하에 상기 버퍼로부터 상기 데이터를 언로드하고,

상기 애플리케이션의 제어하에 상기 언로드된 데이터를 처리하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 11.

청구항 10에 있어서,

상기 제1 및 제2 데이터-전송 오브젝트는 각각 상기 동일한 오브젝트 코드의 제1 및 제2 인스턴스를 포함하는 것을 특징으로 하는 컴퓨팅 머신.

청구항 12.

청구항 10에 있어서,

상기 프로세서는:

상기 애플리케이션을 실행하고 상기 애플리케이션의 제어하에 상기 언로드된 데이터를 처리하도록 동작 가능한 처리 유닛; 및

상기 제1 및 제2 데이터-전송 오브젝트를 실행하고, 상기 버스로부터 상기 데이터를 검색하고, 상기 제1 데이터-전송 오브젝트의 제어하에 상기 버퍼로 상기 데이터를 로드하고, 상기 제2 데이터-전송 오브젝트의 제어하에 상기 버퍼로부터 상기 데이터를 언로드하도록 동작 가능한 데이터-전송 처리기를 포함하는 것을 특징으로 하는 컴퓨팅 머신.

청구항 13.

청구항 10에 있어서,

상기 프로세서는 상기 애플리케이션의 스레드를 실행하고 상기 스레드의 제어하에 상기 언로드된 데이터를 처리하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 14.

청구항 10에 있어서,

뮤 오브젝트 및 판독기 오브젝트를 실행하고,

상기 뮤 오브젝트의 제어하에 상기 발행된 데이터를 상기 제1 버퍼로 로딩하는 것을 반영하는 값인 뮤 값을 저장하고,

상기 판독기 오브젝트의 제어하에 상기 뮤 값을 판독하고,

상기 판독기 오브젝트의 제어하에 및 상기 뮤 값에 응답하여 상기 발행된 데이터가 상기 버퍼를 점유하는 상기 제2 데이터-전송 오브젝트를 통보하고,

상기 제2 데이터-전송 오브젝트의 제어하에 및 상기 통보에 응답하여 상기 버퍼로부터 상기 발행된 데이터를 언로드하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 15.

청구항 10에 있어서,

제2 버퍼를 더 구비하고,

상기 프로세서는 상기 제1 데이터-전송 오브젝트의 제어하에 상기 제2 버퍼로부터 상기 데이터를 검색하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 16.

청구항 10에 있어서,

버스를 더 구비하고,

상기 프로세서는, 통신 오브젝트를 실행하고, 상기 통신 오브젝트의 제어하에 상기 버스로부터 상기 데이터를 수신하고, 상기 제1 데이터-전송 오브젝트의 제어하에 상기 통신 오브젝트로부터 상기 데이터를 검색하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 17.

청구항 10에 있어서,

상기 제1 및 제2 데이터-전송 오브젝트는 각각 상기 동일한 오브젝트 코드의 제1 및 제2 인스턴스를 포함하고,

상기 프로세서는 오브젝트 팩토리를 실행하고 상기 오브젝트 팩토리의 제어하에 상기 오브젝트 코드를 생성하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 18.

청구항 10에 있어서,

상기 프로세서는, 상기 제1 데이터-전송 오브젝트의 제어하에 헤더 및 상기 데이터를 포함하는 메시지로부터 상기 데이터를 복구하도록 동작 가능한 것을 특징으로 하는 컴퓨팅 머신.

청구항 19.

버퍼;

버스;

상기 버퍼 및 상기 버스와 결합된 프로세서; 및

상기 버스와 결합되어 있고 상기 버스로부터 방행된 데이터를 수신하도 상기 수신된 방행된 데이터를 처리하도록 동작 가능한 파이프라인 가속기를 구비하고,

상기 프로세서는,

애플리케이션, 제1 및 제2 데이터-전송 오브젝트, 및 통신 오브젝트를 실행하고,

상기 애플리케이션의 제어하에 데이터를 방행하고,

상기 제1 데이터-전송 오브젝트의 제어하에 상기 방행된 데이터를 상기 버퍼로 로드하고,

상기 제2 데이터-전송 오브젝트의 제어하에 상기 버퍼로부터 상기 방행된 데이터를 검색하고,

상기 통신 오브젝트의 제어하에 상기 버스상으로 상기 방행된 데이터를 구동시키도록 동작 가능한 것을 특징으로 하는 퍼어-백터 머신.

청구항 20.

청구항 19에 있어서,

상기 프로세서는 상기 제2 데이터-전송 오브젝트의 제어하에 상기 방행된 데이터를 포함하는 메시지를 구성하고 상기 통신 오브젝트의 제어하에 상기 버스상으로 상기 메시지를 구동하도록 동작 가능하며,

상기 파이프라인 가속기는 상기 버스로부터 상기 메시지를 수신하고 상기 메시지로부터 상기 방행된 데이터를 복구하도록 동작 가능한 것을 특징으로 하는 퍼어-백터 머신.

청구항 21.

청구항 19에 있어서,

상기 호스트 프로세서에 결합되어 있고 오브젝트 데이터를 저장하도록 동작 가능한 레지스트리를 더 구비하고,

상기 프로세서는,

오브젝트 팩토리를 실행하고,

상기 오브젝트 팩토리의 제어하에 상기 오브젝트 데이터로부터 상기 제1 및 제2 데이터-전송 오브젝트 및 상기 통신 오브젝트를 생성하도록 동작 가능한 것을 특징으로 하는 퍼어-백터 머신.

청구항 22.

버퍼;

버스;

상기 버스에 결합되어 있고 데이터를 생성하고 상기 버스상에 상기 데이터를 구동하도록 동작 가능한 파이프라인 가속기; 및

상기 버퍼 및 상기 버스와 결합되어 있는 프로세서를 구비하고,

상기 프로세서는,

애플리케이션, 제1 및 제2 데이터-전송 오브젝트, 및 통신 오브젝트를 실행하고,

상기 통신 오브젝트의 제어하에 상기 버스로부터 상기 데이터를 수신하고,

상기 제1 데이터-전송 오브젝트의 제어하에 상기 수신된 데이터를 상기 버퍼로 로드하고,

상기 제2 데이터-전송 오브젝트의 제어하에 상기 버퍼로부터 상기 데이터를 언로드하고,

상기 애플리케이션의 제어하에 상기 언로드된 데이터를 처리하도록 동작 가능한 것을 특징으로 하는 피어-백터 머신.

청구항 23.

청구항 22에 있어서,

상기 파이프라인 가속기는 상기 데이터를 포함하는 메시지를 구성하고 상기 버스상에 상기 메시지를 구동하도록 동작 가능하며,

상기 프로세서는,

상기 통신 오브젝트의 제어하에 상기 버스로부터 상기 메시지를 수신하고,

상기 제1 데이터-전송 오브젝트의 제어하에 상기 메시지로부터 상기 데이터를 복구하도록 동작 가능한 것을 특징으로 하는 피어-백터 머신.

청구항 24.

상기 호스트 프로세서와 결합되어 있고 오브젝트 데이터를 저장하도록 동작 가능한 레지스트리를 더 구비하고,

상기 프로세서는,

오브젝트 팩토리를 실행하고,

상기 오브젝트 팩토리의 제어하에 상기 제1 및 제2 데이터-전송 오브젝트 및 상기 통신 오브젝트를 생성하도록 동작 가능한 것을 특징으로 하는 피어-백터 머신.

청구항 25.

제1 버퍼;

버스;

상기 버퍼 및 상기 버스와 결합되어 있는 프로세서; 및

상기 버스와 결합되어 있고 구성 펌웨어를 수신하고 상기 구성 펌웨어를 가지고 자신을 구성하도록 동작 가능한 파이프라인 가속기를 구비하고,

상기 프로세서는,

구성 관리자, 제1 및 제2 데이터-전송 오브젝트, 및 통신 오브젝트를 실행하고,

상기 구성 관리자 및 상기 제1 데이터-전송 오브젝트의 제어하에 상기 버퍼로 구성 펌웨어를 로드하고,

상기 제2 데이터-전송 오브젝트의 제어하에 상기 버퍼로부터 상기 구성 펌웨어를 검색하고,

상기 통신 오브젝트의 제어하에 상기 버스상에 상기 구성 펌웨어를 구동하도록 동작 가능한 것을 특징으로 하는 퍼어-백터 머신.

청구항 26.

청구항 25에 있어서,

상기 프로세서는 상기 제2 데이터-전송 오브젝트의 제어하에 상기 펌웨어를 포함하는 메시지를 구성하고 상기 통신 오브젝트의 제어하에 상기 버스상으로 상기 메시지를 구동하도록 동작 가능하며,

상기 파이프라인 가속기는 상기 버스로부터 상기 메시지를 수신하고 상기 메시지로부터 상기 구성 펌웨어를 복구하도록 동작 가능한 것을 특징으로 하는 퍼어-백터 머신.

청구항 27.

청구항 26에 있어서,

상기 프로세서와 결합되어 있고 구성 데이터를 저장하도록 동작 가능한 레지스트리를 더 구비하고,

상기 프로세서는 상기 구성 관리자의 제어하에 상기 구성 데이터로부터 상기 구성 펌웨어를 위치시키도록 동작 가능한 것을 특징으로 하는 퍼어-백터 머신.

청구항 28.

청구항 25에 있어서,

제2 버퍼를 더 구비하고,

상기 프로세서는,

애플리케이션과 제3 및 제4 데이터-전송 오브젝트를 실행하고,

상기 구성 관리자의 제어하에 구성 명령을 생성하고,

상기 제3 데이터-전송 오브젝트의 제어하에 상기 제2 버퍼로 상기 구성 명령을 로드하고,

상기 제4 데이터-전송 오브젝트의 제어하에 상기 제2 버퍼로부터 상기 구성 명령을 검색하고,

상기 애플리케이션의 제어하에 상기 구성 명령에 대응하는 연산을 수행하도록 상기 애플리케이션을 구성하도록 동작 가능한 것을 특징으로 하는 피어-벡터 머신.

청구항 29.

청구항 25에 있어서,

상기 프로세서는,

상기 구성 관리자의 제어하에 구성 명령을 생성하고,

상기 애플리케이션의 제어하에 상기 구성 명령에 대응하는 연산을 수행하도록 상기 애플리케이션을 구성하도록 동작 가능한 것을 특징으로 하는 피어-벡터 머신.

청구항 30.

청구항 25에 있어서,

상기 구성 관리자는 상기 펌웨어의 로딩 전에 상기 파이프라인 가속기가 상기 구성 데이터에 의해 정의된 구성을 지원하는지 확인하도록 동작 가능한 것을 특징으로 하는 피어-벡터 머신.

청구항 31.

제1 버퍼;

버스;

상기 버스와 결합되어 있고 예외 데이터를 생성하고 상기 버스상에 상기 예외 데이터를 구동하도록 동작 가능한 파이프라인 가속기; 및

상기 버퍼 및 상기 버스와 결합되어 있는 프로세서를 구비하고,

상기 프로세서는,

예외 관리자, 제1 및 제2 데이터-전송 오브젝트 및 통신 오브젝트를 실행하고,

상기 통신 오브젝트의 제어하에 상기 버스로부터 상기 예외 데이터를 수신하고,

상기 제1 데이터-전송 오브젝트의 제어하에 상기 버퍼로 상기 수신된 예외 데이터를 로드하고,

상기 제2 데이터-전송 오브젝트의 제어하에 상기 버퍼로부터 상기 예외 데이터를 언로드하고,

상기 예외 관리자의 제어하에 상기 언로드된 예외 데이터를 처리하도록 동작 가능한 것을 특징으로 하는 피어-벡터 머신.

청구항 32.

청구항 31에 있어서,

상기 파이프라인은 상기 예외 데이터를 포함하는 메시지를 구성하고 상기 메시지를 상기 버스상으로 구동시키도록 동작 가능하며,

상기 프로세서는 상기 통신 오브젝트의 제어하여 상기 버스로부터 상기 메시지를 수신하고, 상기 제1 데이터-전송 오브젝트의 제어하에 상기 메시지로부터 상기 예외 데이터를 복구하도록 동작 가능한 것을 특징으로 하는 피어-백터 머신.

청구항 33.

제2 버퍼를 더 구비하고,

상기 프로세서는,

상기 구성 관리자와 상기 제3 및 제4 데이터-전송 오브젝트를 실행하고,

상기 예외 데이터에 응답하여 상기 구성 관리자의 제어하에 구성 펌웨어를 생성하고,

상기 제3 데이터-전송 오브젝트의 제어하에 상기 구성 펌웨어를 상기 제2 버퍼로 로드하고,

상기 제4 데이터-전송 오브젝트의 제어하에 상기 제2 버퍼로부터 상기 구성 명령을 언로드하고,

상기 통신 오브젝트의 제어하에 상기 구성 펌웨어를 상기 버스상으로 구동시키도록 동작 가능하며,

상기 파이프라인 가속기는 상기 버스로부터 상기 구성 펌웨어를 수신하고 상기 펌웨어를 가지고 자기 자신을 구성하도록 동작 가능한 것을 특징으로 하는 피어-백터 머신.

청구항 34.

청구항 31에 있어서,

상기 프로세서는,

애플리케이션 및 구성 관리자를 실행하고,

상기 예외 데이터에 응답하여 상기 구성 관리자의 제어하에 구성 명령을 생성하고,

상기 구성 명령에 응답하여 상기 애플리케이션의 제어하에 상기 애플리케이션을 재구성하도록 동작 가능한 것을 특징으로 하는 피어-백터 머신.

청구항 35.

구성 데이터를 저장하도록 동작 가능한 구성 레지스트리;

상기 구성 레지스트리와 결합되어 있고 상기 구성 데이터로부터 구성 펌웨어를 얻어지키도록 동작 가능한 프로세서; 및

상기 프로세서와 결합되어 있고 상기 구성 펌웨어를 가지고 자기 자신을 구성하도록 동작 가능한 파이프라인 가속기를 구비하는 것을 특징으로 하는 피어-백터 머신.

청구항 36.

구성 데이터를 저장하도록 동작 가능한 구성 레지스트리;

파이프라인 가속기; 및

상기 구성 레지스트리 및 상기 파이프라인 가속기와 결합되어 있고, 상기 구성 데이터에 응답하여 구성 펌웨어를 수신하고 상기 구성 펌웨어를 가지고 상이 파이프라인 가속기를 구성하도록 동작 가능한 것을 특징으로 하는 퍼어-벡터 머신.

청구항 37.

애플리케이션이 있는 데이터를 발행하는 단계;

제1 데이터-전송 오브젝트를 가지고 상기 발행된 데이터를 제1 버퍼로 로드하는 단계; 및

제2 데이터-전송 오브젝트를 가지고 상기 버퍼로부터 상기 발행된 데이터를 검색하는 단계를 구비하는 것을 특징으로 하는 방법.

청구항 38.

청구항 37에 있어서,

상기 발행된 데이터는 상기 애플리케이션의 스레드가 있는 데이터를 발행하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 39.

청구항 37에 있어서,

상기 버퍼내에 상기 발행된 데이터의 존재에 대응하는 큐 값을 생성하는 단계; 및

상기 큐 값에 응답하여 상기 발행된 데이터가 상기 버퍼를 점유하는 상기 제2 데이터-전송 오브젝트를 통보하는 단계를 더 구비하고,

상기 발행된 데이터는 상기 통보에 응답하여 상기 제2 데이터-전송 오브젝트를 가지고 상기 저장 위치로부터 상기 발행된 데이터를 검색하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 40.

청구항 37에 있어서,

통신 오브젝트를 가지고 버스상에 상기 검색된 데이터를 구동시키는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 41.

청구항 37에 있어서,

상기 제2 데이터-전송 오브젝트를 가지고 제2 버퍼로 상기 검색된 데이터를 로드하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 42.

청구항 37에 있어서,

상기 제2 데이터-전송 오브젝트를 가지고 상기 검색된 데이터를 위한 헤더를 생성하는 단계; 및

상기 제2 데이터-전송 오브젝트를 가지고 상기 헤더와 상기 검색된 데이터를 메시지와 결합하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 43.

청구항 37에 있어서,

오브젝트 팩토리를 가지고 데이터-전송 오브젝트 코드를 생성하는 단계;

상기 오브젝트 코드의 제1 인스턴스로서 상기 제1 데이터-전송 오브젝트를 생성하는 단계; 및

상기 오브젝트 코드의 제2 인스턴스로서 상기 제2 데이터-전송 오브젝트를 생성하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 44.

청구항 37에 있어서,

파이프라인 가속기를 가지고 상기 제2 데이터-전송 오브젝트로부터 상기 데이터를 수신 및 처리하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 45.

데이터를 검색하고, 상기 검색된 데이터를 제1 데이터-전송 오브젝트를 가지고 제1 버퍼로 로드하는 단계;

제2 데이터-전송 오브젝트를 가지고 상기 버퍼로부터 상기 데이터를 언로드하는 단계; 및

애플리케이션을 가지고 상기 언로드된 데이터를 처리하는 단계를 구비하는 것을 특징으로 하는 방법.

청구항 46.

청구항 45에 있어서,

상기 언로드된 데이터를 처리하는 단계는, 상기 애플리케이션의 스레드를 가지고 상기 언로드된 데이터를 처리하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 47.

청구항 45에 있어서,

상기 버퍼내의 상기 데이터의 존재에 대응하는 ㅠ 값을 생성하는 단계; 및

상기 ㅠ 값에 응답하여 상기 데이터가 상기 버퍼를 점유하고 있음을 상기 제2 데이터-전송 오브젝트로 통보하는 단계를 더 구비하고,

상기 데이터를 언로드하는 단계는, 상기 통보에 응답하여 상기 제1 데이터-전송 오브젝트를 가지고 상기 버퍼로부터 상기 데이터를 언로드하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 48.

청구항 45에 있어서,

상기 데이터를 검색하는 단계는, 상기 제1 데이터-전송 오브젝트를 가지고 제2 버퍼로부터 상기 데이터를 검색하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 49.

청구항 45에 있어서,

통신 오브젝트를 가지고 버스로부터 상기 데이터를 수신하는 단계를 더 구비하고,

상기 데이터를 검색하는 단계는, 상기 제1 데이터-전송 오브젝트를 가지고 상기 통신 오브젝트로부터 상기 데이터를 검색하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 50.

청구항 45에 있어서,

파이프라인 가속기를 가지고 상기 데이터를 상기 제1 데이터-전송 오브젝트로 제공하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 51.

프로세서상에서 구동하는 애플리케이션이 있는 데이터를 발행하는 단계;

상기 프로세서상에서 구동하는 제1 데이터-전송 오브젝트를 가지고 상기 발행된 데이터를 버퍼로 로드하는 단계;

상기 프로세서상에서 구동하는 제2 데이터-전송 오브젝트를 가지고 상기 발행된 데이터를 상기 버퍼로부터 검색하는 단계;

상기 프로세서상에서 구동하는 통신 오브젝트를 가지고 상기 검색된 발행된 데이터를 버스상에서 구동시키는 단계; 및

파이프라인 가속기를 가지고 상기 버스로부터 상기 방행된 데이터를 수신하고 상기 방행된 데이터를 처리하는 단계를 구비하는 것을 특징으로 하는 방법.

청구항 52.

청구항 51에 있어서,

상기 제2 데이터-전송 오브젝트를 가지고 헤더 및 상기 방행된 데이터를 포함하는 메시지를 생성하는 단계를 더 구비하고,

상기 버스상에 상기 데이터를 구동시키는 단계는, 상기 통신 오브젝트를 가지고 상기 메시지를 상기 버스상에 구동시키는 단계를 포함하고,

상기 방행된 데이터를 수신 및 처리하는 단계는, 상기 파이프라인 가속기를 가지고 상기 메시지로부터 상기 메시지를 수신하고 상기 방행된 데이터를 복구하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 53.

데이터를 생성하고, 파이프라인 가속기를 가지고 상기 데이터를 버스상에 4구동시키는 단계;

상기 통신 오브젝트를 가지고 상기 버스로부터 상기 데이터를 수신하는 단계;

제1 데이터-전송 오브젝트를 가지고 상기 수신된 데이터를 버퍼로 로드하는 단계;

제2 데이터-전송 오브젝트를 가지고 상기 버퍼로부터 상기 데이터를 언로드하는 단계; 및

애플리케이션을 가지고 상기 언로드된 데이터를 처리하는 단계를 구비하는 것을 특징으로 하는 방법.

청구항 54.

청구항 53에 있어서,

상기 데이터를 생성하는 단계는, 상기 파이프라인 가속기를 가지고 헤더 및 상기 데이터를 포함하는 메시지를 구성하는 단계를 포함하고,

상기 데이터를 구동하는 단계는, 상기 파이프라인 가속기를 가지고 상기 메시지를 상기 버스상에서 구동시키는 단계를 포함하고,

상기 데이터를 수신하는 단계는, 상기 통신 오브젝트를 가지고 상기 버스로부터 상기 메시지를 수신하는 단계를 포함하며,

상기 제1 데이터-전송 오브젝트를 가지고 상기 메시지로부터 상기 데이터를 복구하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 55.

구성 관리자를 가지고 구성 펌웨어를 검색하는 단계;

제1 통신 오브젝트를 가지고 상기 구성 펌웨어를 제1 버퍼로 로드하는 단계;

제2 통신 오브젝트를 가지고 상기 버퍼로부터 상기 구성 펌웨어를 검색하는 단계;

통신 오브젝트를 가지고 상기 구성 펌웨어를 버스상에 구동시키는 단계;

파이프라인 가속기를 가지고 상기 구성 펌웨어를 수신하는 단계; 및

상기 구성 펌웨어를 가지고 상기 파이프라인 가속기를 구성하는 단계를 구비하는 것을 특징으로 하는 방법.

청구항 56.

청구항 55에 있어서,

상기 구성 관리자를 가지고 구성 명령을 생성하는 단계; 및

상기 구성 명령에 대응하는 연산을 수행하도록 상기 애플리케이션을 구성하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 57.

청구항 55에 있어서,

상기 구성 관리자를 가지고 구성 명령을 생성하는 단계;

제3 통신 오브젝트를 가지고 상기 구성 명령을 제2 버퍼로 로드하는 단계;

제4 통신 오브젝트를 가지고 상기 제2 버퍼로부터 상기 구성 명령을 검색하는 단계; 및

상기 구성 명령에 대응하는 연산을 수행하도록 상기 애플리케이션을 구성하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 58.

예외 데이터를 생성하고, 파이프라인 가속기를 가지고 상기 예외 데이터를 버스상에서 구동시키는 단계;

통신 오브젝트를 가지고 상기 버스로부터 상기 예외 데이터를 수신하는 단계;

제1 데이터-전송 오브젝트를 가지고 상기 수신된 예외 데이터를 버퍼로 로드하는 단계;

제2 데이터-전송 오브젝트를 가지고 상기 버퍼로부터 상기 예외 데이터를 언로드하는 단계; 및

예외 관리자를 가지고 상기 언로드된 예외 데이터를 처리하는 단계를 구비하는 것을 특징으로 하는 방법.

청구항 59.

청구항 58에 있어서,

상기 에러 데이터에 응답하여 구성 관리자를 가지고 구성 펌웨어를 검색하는 단계;

제3 전송 오브젝트를 가지고 상기 구성 펌웨어를 제2 버퍼로 로드하는 단계;

제4 데이터-전송 오브젝트를 가지고 상기 제2 버퍼로부터 상기 구성 명령을 인로드하는 단계;

상기 통신 오브젝트를 가지고 상기 구성 펌웨어를 상기 버스상에 구동시키는 단계; 및

상기 구성 펌웨어를 가지고 상기 파이프라인 가속기를 재구성하는 단계를 더 구비하는 것을 특징으로 하는 방법.

청구항 60.

청구항 58에 있어서,

상기 에러 데이터에 응답하여 구성 관리자를 가지고 구성 명령을 생성하는 단계; 및

상기 구성 명령에 응답하여 상기 애플리케이션을 재구성하는 단계를 더 구비하는 것을 특징으로 하는 방법.

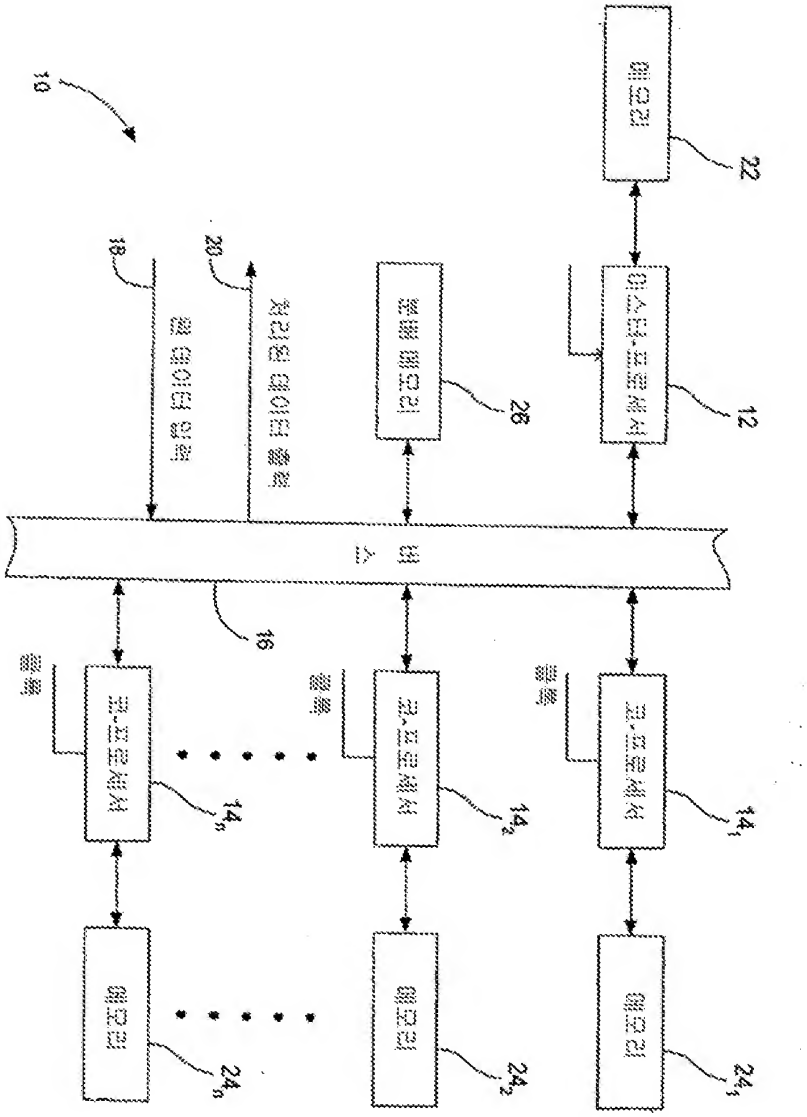
청구항 61.

컴퓨팅 머신의 초기화 동안에 구성 레지스트리에 저장된 구성 데이터에 의해 지정된 구성 펌웨어를 검색하는 단계; 및

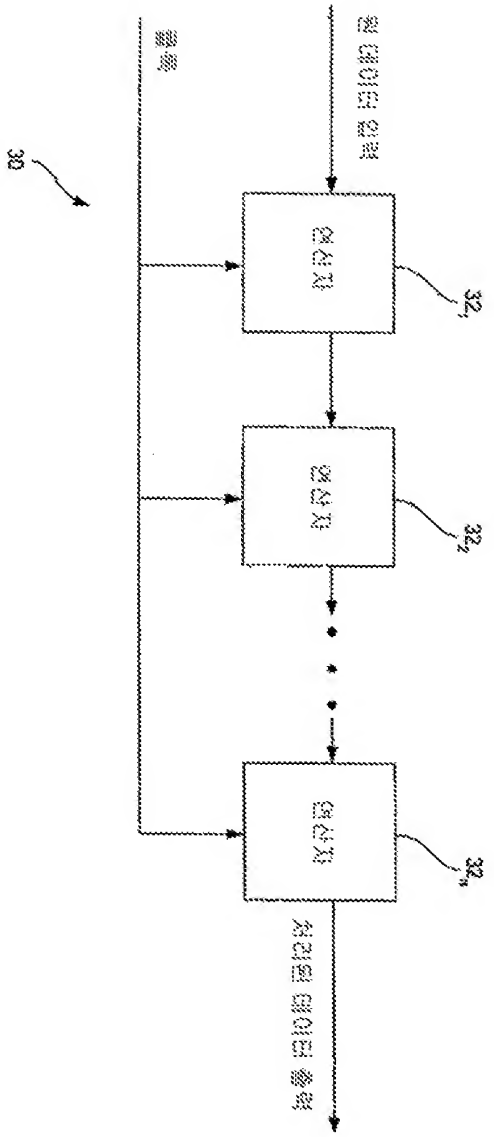
상기 구성 펌웨어를 가지고 상기 컴퓨팅 머신의 파이프라인 가속기를 구성하는 단계를 구비하는 것을 특징으로 하는 방법.

도면

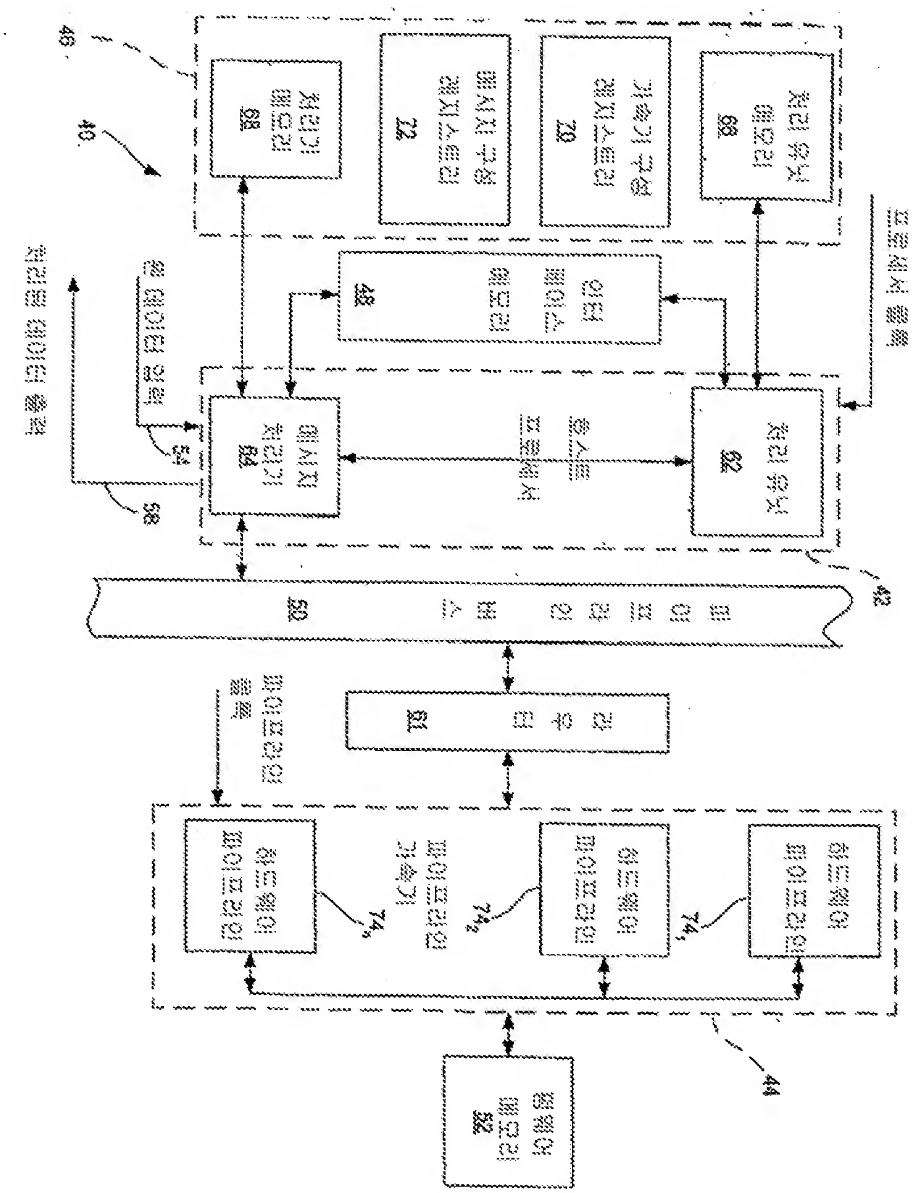
도면1


10
22
메모리
12
마스터-프로세서
26
보조 메모리
18
입력 데이터
20
처리된 데이터 출력
버스
16
클록
코-프로세서
14_1
14_2
14_n
24_1
24_2
24_n

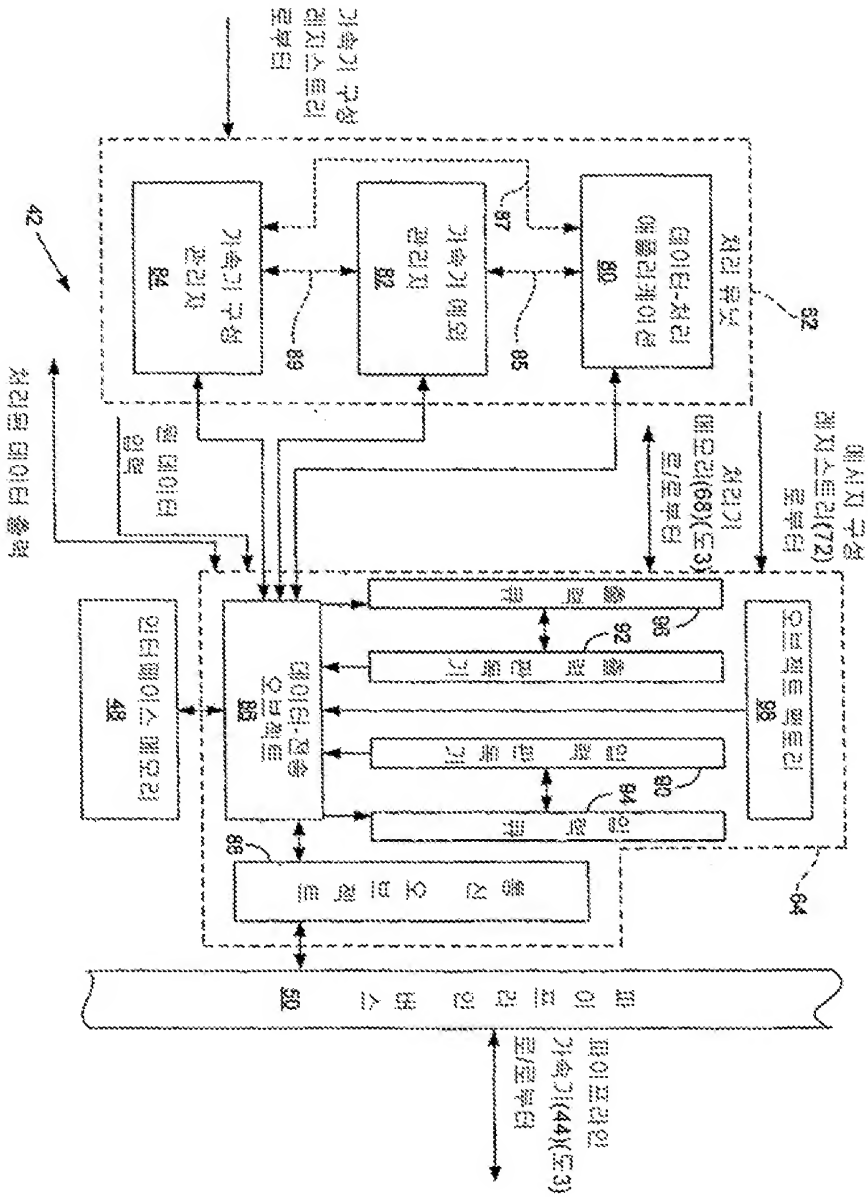
도면2



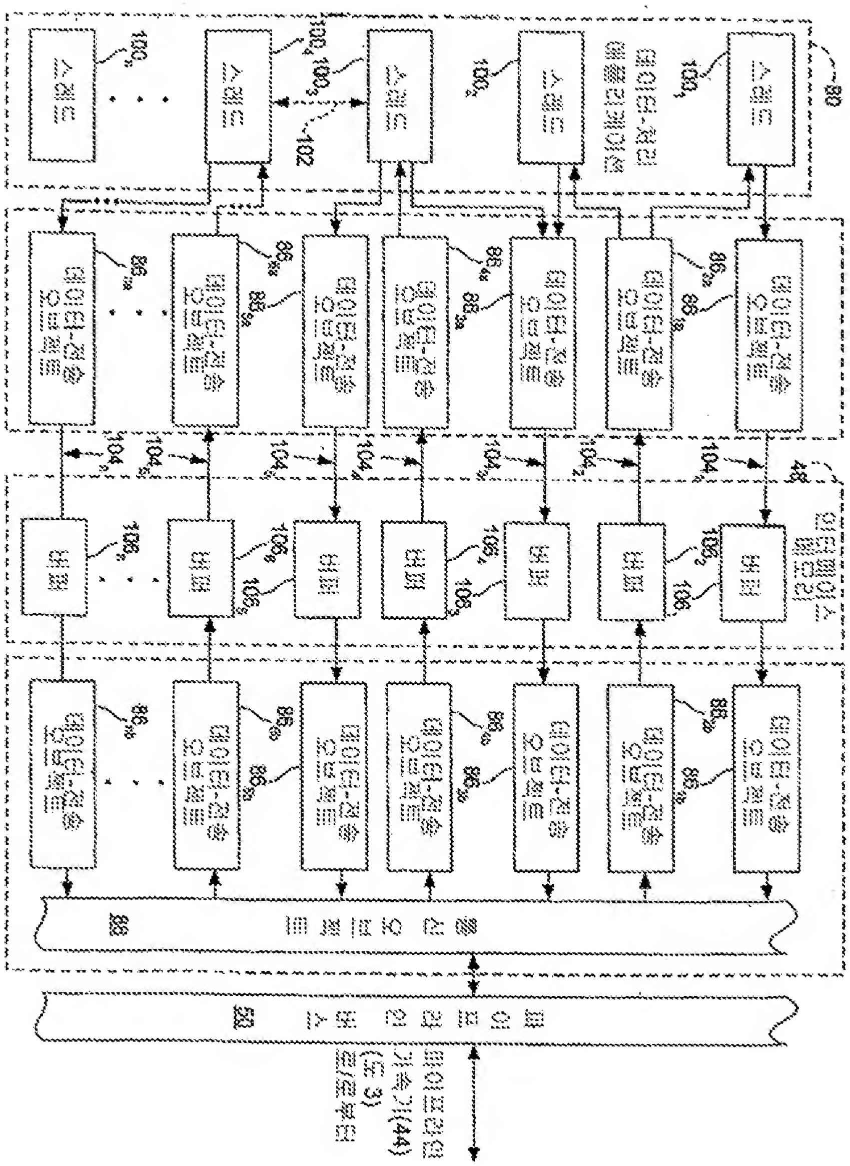
도면3


프로세서 출력
42
처리 유닛 메모리
66
처리 유닛
62
가속기 구성 레지스트리
70
인터페이스 메모리
48
호스트 프로세서
메시지 구성 레지스트리
72
처리기 메모리
68
메시지 처리기
64
54
원 데이터 입력
58
처리된 데이터 출력
46
40
파이프라인 버스
50
라우터
61
파이프라인 출력
하드웨어 파이프라인
74_1
하드웨어 파이프라인
74_2
파이프라인 가속기
74_n
하드웨어 파이프라인
44
펌웨어 메모리
52

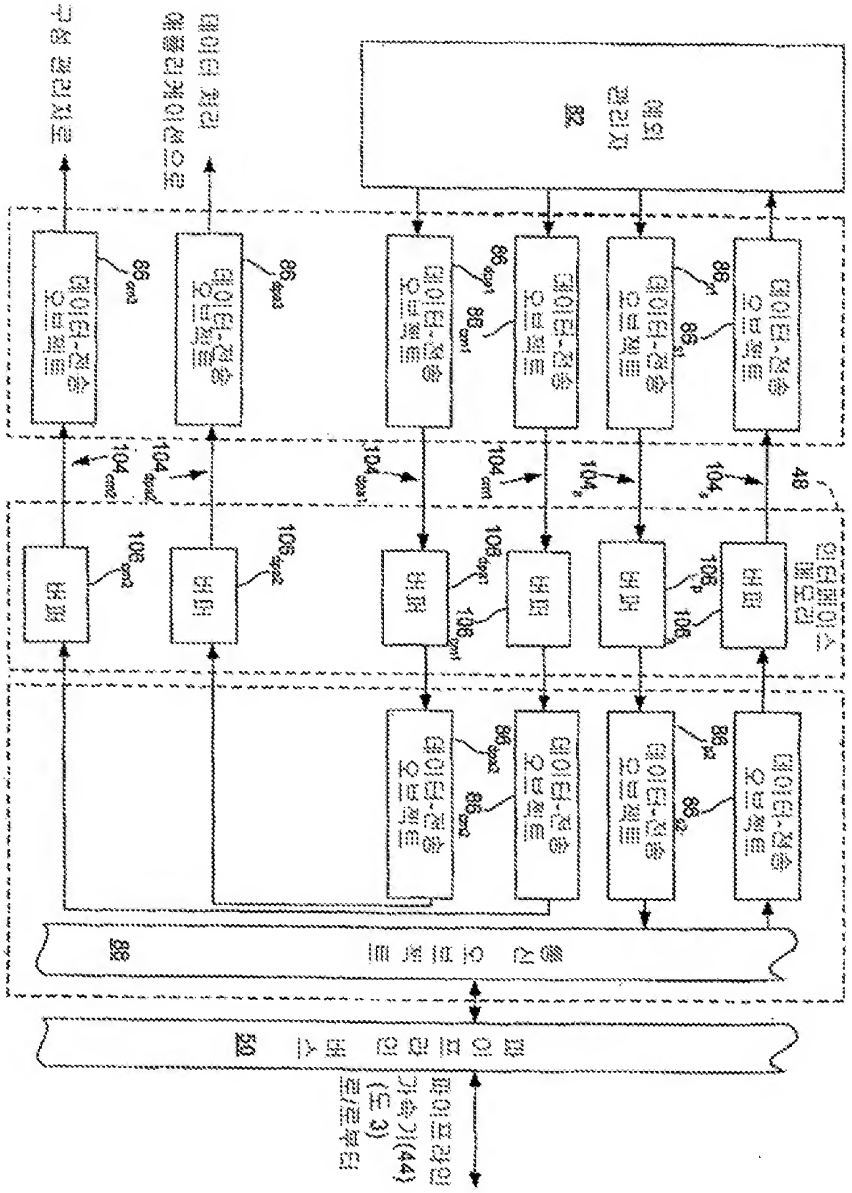
도면4


가속기 구성 레지스터 클록 타임
42
처리할 데이터 출력
입출 데이터 입출
메시지 구성 레지스터2(72) 클록 타임
처리기 데이터2(68)(도3) 클록 타임
82
처리 유닛
데이터-처리 레퍼런스 80
가속기 레퍼런스 관리자 82
가속기 구성 관리자 84
87
85
89
64
인터페이스 레지스터 48
데이터-전송 오브젝트 85
오브젝트 레퍼런스 86
제1 레지스터 90
제2 레지스터 92
제3 레지스터 94
제4 레지스터 96
버스 인터페이스 유닛 50
버스 인터페이스 클록 타임
가속기(44)(도3) 클록 타임

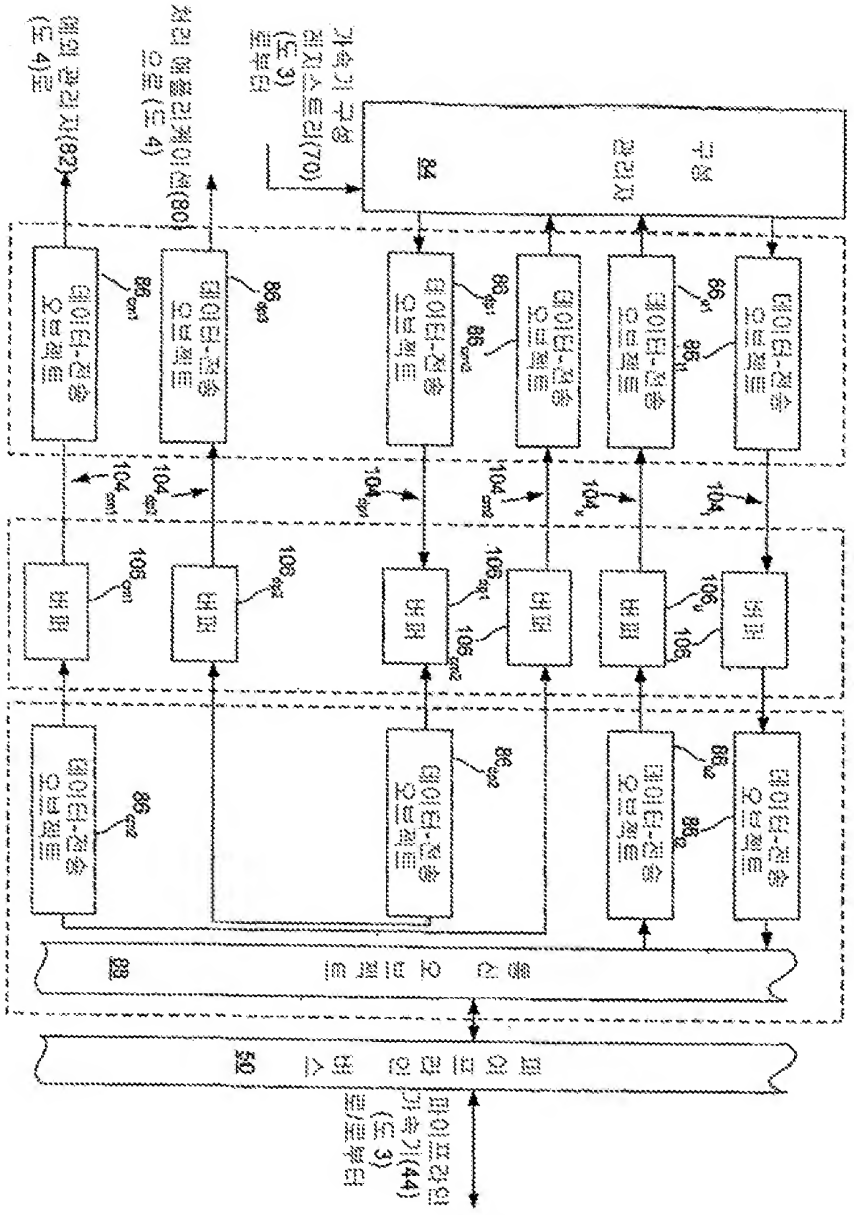
도면6



도면6


메모리 클라이언트
82
데이터-정송 오브젝트
데이터-정송 오브젝트
데이터-정송 오브젝트
데이터-정송 오브젝트
데이터-정송 오브젝트
데이터-정송 오브젝트
데이터-정송 오브젝트
데이터-정송 오브젝트
버퍼
버퍼
버퍼
버퍼
버퍼
버퍼
버퍼
버퍼
48
88
50
데이터 처리 애플리케이션으로
구성 관리자로
패이프라인 기술자(44) (도 3) 로/로부터

도면7


제어 로직 지연(82)
처리 엘리먼트 인에이블(80)
기록 리드 제어 신호(70)
(도 3)
출력 타임
(도 4)
(도 4)
제어기
84
데이터-경로 오브젝트
86_{ext}
86_{out1}
86_{out2}
86_{in1}
86_{in2}
86_{cm1}
86_{cm2}
104_{ext}
104_{out1}
104_{out2}
104_{in1}
104_{in2}
104_{cm1}
104_{cm2}
버퍼
106_{ext}
106_{out1}
106_{out2}
106_{in1}
106_{in2}
106_{cm1}
106_{cm2}
88
50
데이터 입력 버스
데이터 출력 버스
데이터 제어 신호(44)
(도 3)
출력 타임